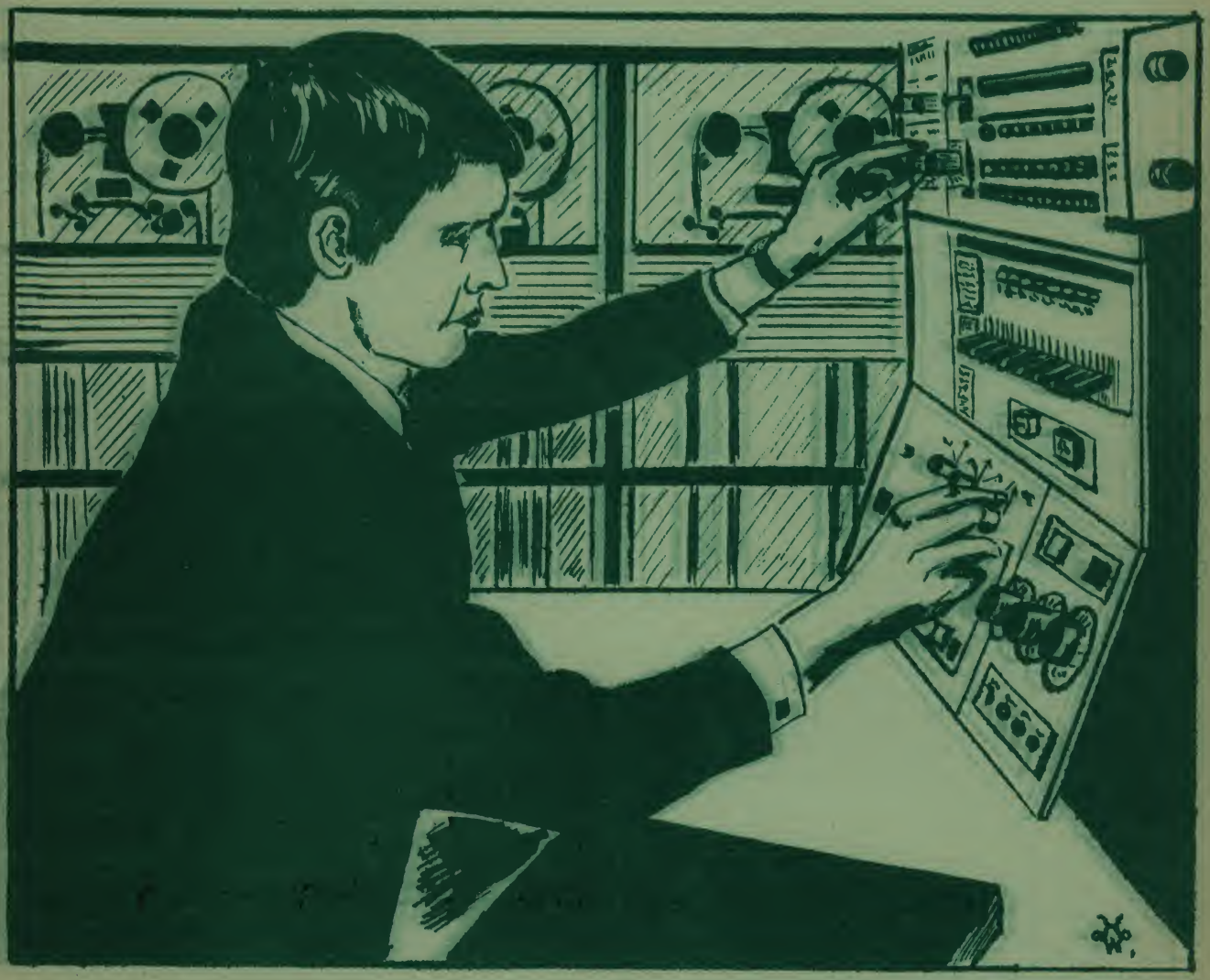
E C S

COMPUTER PROGRAMMEUR



Les 3

H. Wenteler

1. INLEIDING

De problemen die op een rekenmachine worden opgelost bestaan uit een serie achtereenvolgende stappen die men instrukties noemt. Bij de SERA zijn dit maximaal 100 instrukties. Instrukties die over het algemeen niet erg moeilijk zijn, evenmin als de handelingen die door de instrukties verricht moeten worden. De moeilijkheidsfaktor in het programmeren schuilt dan ook meestal in de juiste organisatie. M.a.w. de instrukties in de juiste volgorde te plaatsen. Dit probleem is des te groter wanneer het aantal te gebruiken instrukties van ruime omvang is.

Een probleem kan uit zoveel instrukties gaan bestaan, dat het praktisch onmogelijk wordt, zonder hulpmiddelen een goed programma te ontwerpen.

MEERDERE FASEN

Het is over het algemeen niet mogelijk dat men onmiddelijk begint aan het programmeren van een probleem. Vooraf zal men reeds in grote lijnen een oplossing voor dat probleem gevonden moeten hebben. Bovendien zal blijken dat het betreffende probleem vaak slechts een onderdeel vormt van een veel groter geheel. Wij zullen U dat iets nader toelichten.

Het kan in een administratie voorkomen dat een bepaald administratief objekt een onderdeel is van een veel groter automatiseringsprojekt. Zo kan men denken aan een bedrijf dat ertoe overgaat om de voorraadadministratie te automatiseren. Voor dit voorraadprobleem zal een programma worden geschreven, maar dit leidt tot meerdere gevolgen. Er zal **een** andere wijze van voorraadadministratie ontstaan dan er in het verleden plaats vond. Bepaalde informatie was vroeger niet beschikbaar, terwijl deze nu wel leverbaar is. Het omgekeerde geval kan ook voorkomen. Een simpel probleem doet zich al gelijk voor als we er aan denken hoe de computer aan de gegevens komt die de voorraad wijzigen. Of, hoe bewaart de computer de oude voorraad?

ER ONTSTAAT EEN INFORMATIESTROOM

Het gehele bedrijf komt nu in aanraking met de veranderingen. Men zal nu zorgvuldig moeten overwegen hoe de verschillende informatiestromen zullen gaan lopen. Zo zal men zich af moeten vragen :

- *waar ontstaat informatie*
- *in welke vorm komt die informatie bij de diverse personen of afdelingen*
- *welke bewerkingen ondergaat de informatie*
- *in welke vorm dient de informatie aan de machine toegevoerd te worden*
- *in welke vorm komt de informatie uit de machine*
- *voor wie is de informatie bestemd.*

U ziet, er is een "stroom" van informatie ontstaan die zich door het bedrijf begeeft.

SYSTEEM-ANALIST
Een persoon die zich bezig houdt met het bestuderen (het analyseren) van de huidige methode en een oplossing bedenkt om het probleem te automatiseren, noemt men de systeem-analist. Soms, vooral in grote bedrijven wordt het analyseren door de systeem-analist gedaan, terwijl een andere persoon de nieuwe automatiserings methode ontwikkeld. In een dergelijk geval spreekt men van de *SYSTEEM-ONTWERPER*.

De programmeur ontvangt van de systeem-analist de nodige gegevens teneinde zijn programma te kunnen schrijven. Die gegevens worden dan veelal in schemavorm verstrekt. Hier komen we nu aan het begrip stroomschema.

N.B. Vooral bij kleinere computersystemen vindt men de funktie van systeem-analist en programmeur verenigd in één persoon.

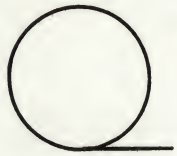
2.1 PRESENTATIE D.M.V. EEN SCHEMA

Het is mogelijk dat een systeem-analist een probleem vastlegt in een rapport. Dit is echter vrij onoverzichtelijk. Beter is het een probleem vast te leggen in de vorm van een schema, waarbij diverse symbolen gebruikt worden die ieder voor zich een eigen betekenis hebben. Enkele van deze symbolen zien er als volgt uit :

Dokument. Alle soorten formulieren, overzichten dokumenten, enz., gebruikt als invoergegevens of vervaardigd tijdens de verwerking.

Ponskaart of alle in- en uitvoer met behulp van ponskaarten.

Ponsband of alle in- en uitvoer met behulp van ponsband.

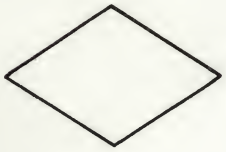


Magneetband of alle in- en uitvoer met behulp van magneetband.

Magneetschijf. Voor het aangeven van bewerkingen waarbij gebruik gemaakt wordt van magneetschijven.

Algemeen basissymbool. Wordt over het algemeen gebruikt om een hoofdbewerking aan te geven.

Verbindingslijn. Hiermede wordt de volgorde van de bewerkingen aangegeven. De normale volgorde is van boven naar beneden, en van links naar rechts. Indien wordt afgeweken van deze normale volgorde, dan worden de verbindingslijnen van een pijl voorzien.



Beslissingssymbool. Symbool voor een vraagstelling en de naar aanleiding daarvan te nemen beslissing.

Connector. Het symbool dat de ingang of de uitgang van een tak van het programma op hetzelfde blad aangeeft.

Offpage connector. Dezelfde betekenis als het voorgaande symbool, echter toegepast op aansluitingen op een ander blad.

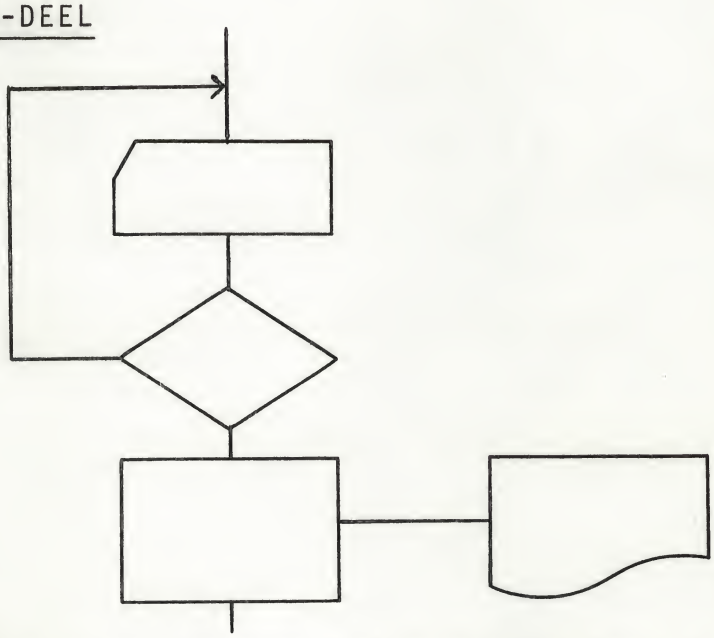
2.2. DIVERSE SCHEMAVORMEN

Alvorens we enkele schema's zullen uitwerken bespreken we eerst nog enige schemavormen.

HET ORGANISATIESCHEMA

Het organisatieschema is een totaalschema van de diverse doorgangen van het projekt, in verwerkingsvolgorde getekend en wel zo, dat een uitvoer-symbool direkt als invoer-symbool van de volgende verwerking kan plaats vinden. Daar er na het organisatieschema nog andere schema's volgen zoals *HOOFDBLOKSCHEMA* en *DETAILBLOKSCHEMA*, is het noodzakelijk elke run van het organisatieschema een unieke naam te geven, die ook in de volgende schema's gebruikt dient te worden ! Zo zullen ook alle bestanden van vaste namen voorzien moeten worden.

VOORBEELD SCHEMA-DEEL



HOOFDBLOKSCHEMA

Een eerste stap in de analyse van elke machinefase is de inventarisatie van de hoofdbewerkingen welke de machine in de fase achtereenvolgens moet realiseren. Deze hoofdbewerkingen, samengebracht in een schema waaruit tevens de interne machine-organisatie van de desbetreffende fase blijkt, wordt het hoofdblokschema genoemd.

DETAILBLOKSCHEMA

Elke verwerking in een fase valt uiteen in afzonderlijke eenheden, welke in de juiste volgorde gerealiseerd, het gewenste resultaat leveren.
Een mutatieverwerking van een hoofdbestand kan bijv. uit de volgende hoofdbewerkingen bestaan :

- *programma start*
- *lees record van oud bestand*
- *lees mutatierecord*
- *indien mutatierecord gelijk is aan record oud bestand, volgt mutatie*

- *gemuteerde record na alle desbetreffende mutaties printen*
- *wegschrijven gemuteerde en niet gemuteerde records naar nieuw bestand*
- *extra routine voor mutaties waarbij geen bestand-record aanwezig is*
- *eindroutine*

In het detailblokschema zal van elk hiervoor genoemd hoofdblokschema een detailschema volgen.

2.3 TOE TE PASSEN WERKWIJZE

De juiste werkwijze bij het in schema brengen van een probleem is, eerste trachten de verwerking in logische hoofdbewerkingen te splitsen.
Deze hoofdbewerkingen worden vervolgens zodanig in schemavorm vastgelegd, dat tevens hun onderlinge relatie duidelijk blijkt. Vanuit de aldus ontstane hoofdblokschema's worden er dan detailblokschema's opgesteld.

TOELICHTEND RAPPORT

Vooral bij overdracht van de hoofdschema's aan andere funktionarissen is een begeleidend rapport een noodzaak.

AAN HET BLOKSCHEMA TE STELLEN EISEN

Aan het blokschema dienen wel eisen gesteld te worden.
Eisen die wij als volgt hebben samengevat :

- *het blokschema moet op het probleem georiënteerd zijn.*
- *het blokschema is de vastlegging van de verwerking in een fase, zijnde een deel van het organisatie-schema, en geeft als zodanig de analyse weer van het probleem in die fase*
- *het blokschema moet overzichtelijk zijn*
- *het blokschema moet eenduidig zijn. Een belangrijke bijdrage hiertoe is het opstellen van een lijst van symbolische namen*
- *het blokschema moet voor iedereen duidelijk zijn. Het mag geen vakjargon en geen onbegrijpelijke afkortingen bevatten.*

Het is van groot belang dat ook de niet-computerdeskundigen, na korte uitleg, het blokschema kunnen begrijpen en kunnen kontroleren. Dit laatste is weer van groot belang voor de "materie"-deskundigen (de div. afd. chefs) Hierdoor worden de diverse afdelingsleiders nauwer bij het projekt betrokken.

2.4 SUBROUTINES

Zonder nog te diep op het begrip subroutine in te gaan is het toch belangrijk reeds nu een zeer beknopte uitleg te geven. In de schema's zal het gebruik van de subroutines en wissels (die hierna aan de orde komen) veelvuldiger aan de orde zijn.

Definitie. Een subroutine is een reeks bewerkingen,welke op diverse punten in het schema opgeroepen en uitgevoerd kan worden.

Voor subroutines wordt dezelfde schematechniek gebruikt.

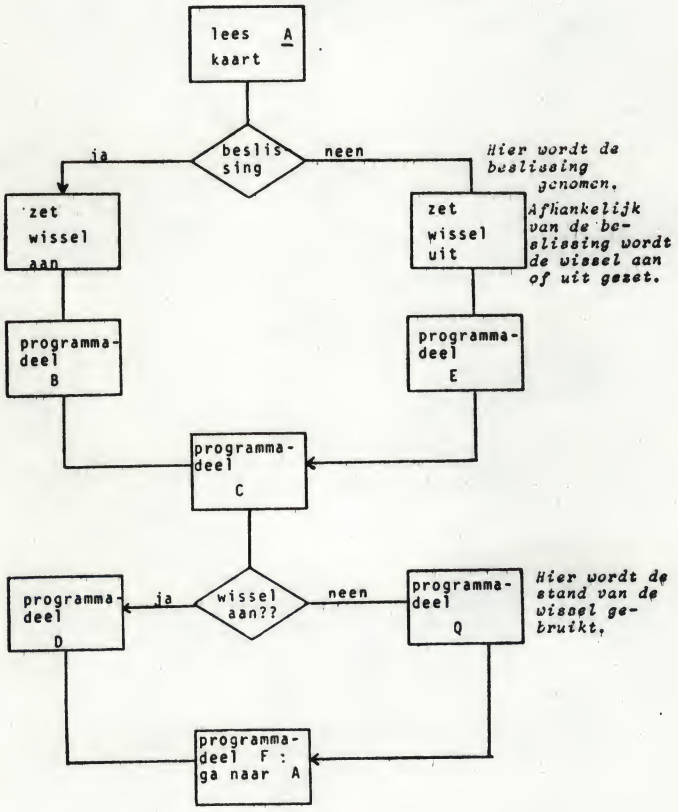
GESLOTEN SUBROUTINE
Dit is een op zichzelf staande routine (apart programma) die op meerdere plaatsen in het programma wordt opgeroepen.

OPEN SUBROUTINE
Hierbij wordt de routine bij het schrijven van het programma mee geschreven en doet dan ook alleen dienst voor dat ene programma.

WISSELS *andere benaming is INDICATOR.*

Definitie. Een wissel is een inrichting, die geprogrammeerd wordt om een bepaalde , eerder genomen beslissing vast te leggen, teneinde deze later weer te kunnen gebruiken.

Voorbeeld. Een wissel dient om te bepalen of op een bepaald moment de programma-delen B, C en D of E, C en Q gebruikt moet worden.



H. Wenteler

3. SCHEMA VOORBEELDEN

Voorbeeld 1.
Voor de uitvoering van de fakturering heeft een onderneming de beschikking over :
a. Naam, Adres, Woonplaats kaarten (dit zijn ponskaarten)
b. orderkaarten

In de N.A.W. kaarten zijn per debiteur opgenomen : het deb. nr., plus naam/adres/woonplaats.

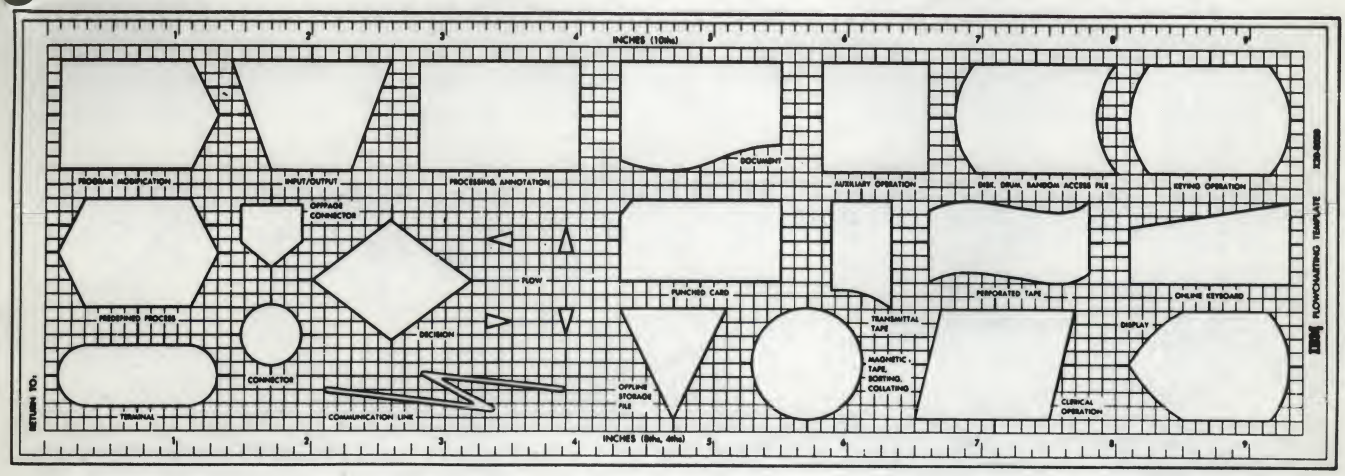
In de orderkaarten zijn opgenomen :
1. het bestelde art. nr.
2. de art. omschrijving
3. besteld aantal
4. prijs per eenheid
5. het verschuldigde bedrag

Er wordt van uit gegaan dat bij elke N.A.W. kaart, één of meerdere orderkaarten aanwezig zijn.

Iedere faktuur moet via het programma van een datum worden voorzien. Deze datum wordt ingelezen via een constante-kaart.

Een uitwerking van dit probleem ziet U op de volgende pagina's in de vorm van een hoofdblokschema en enkele detail-blokschema's.

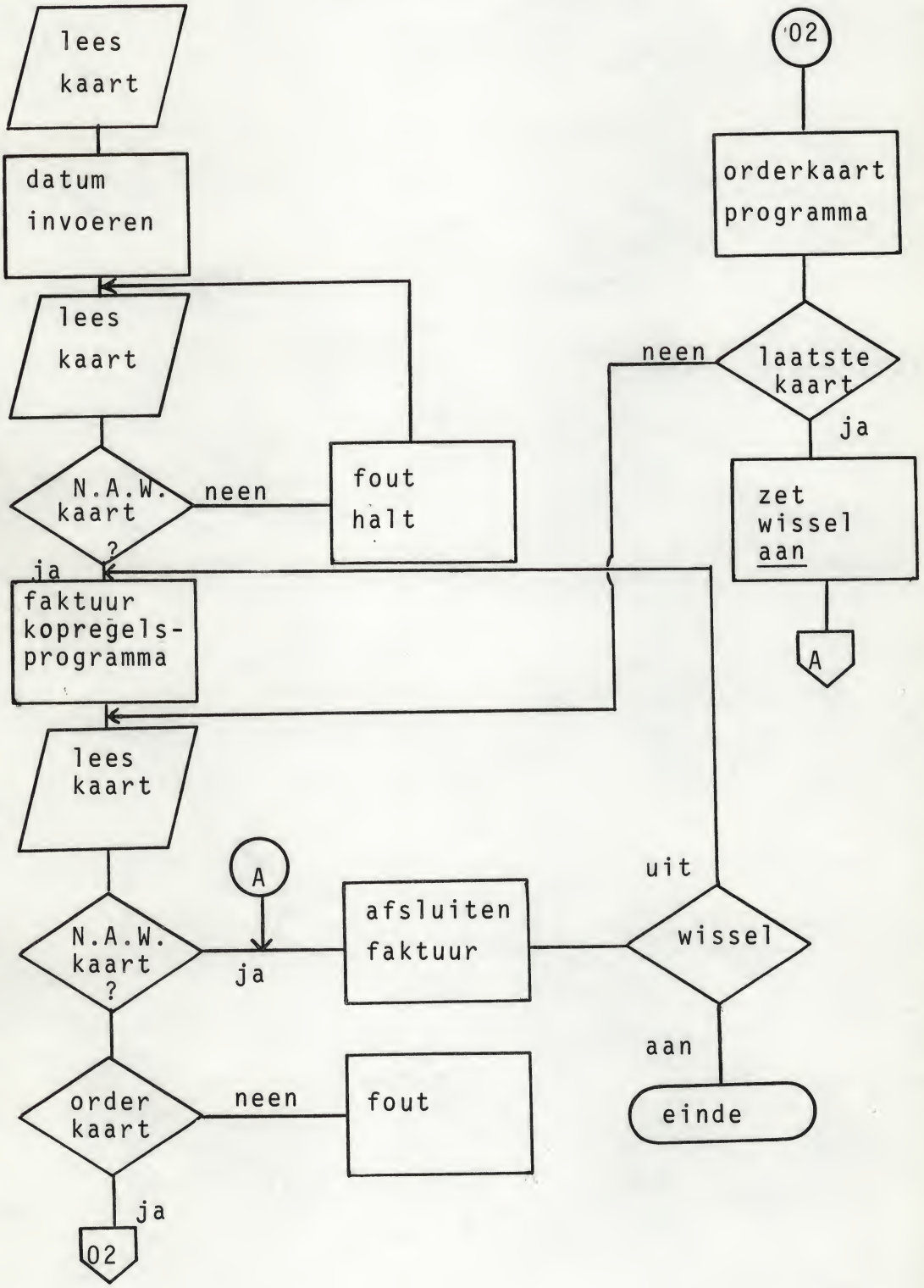
Misschien zijn enkele gegevens in de symbolen U nog niet duidelijk, dat is begrijpelijk.
Het gaat ons hier echter in de eerste plaats om U een globaal inzicht te geven in het doel en het nut van stroomschema's.



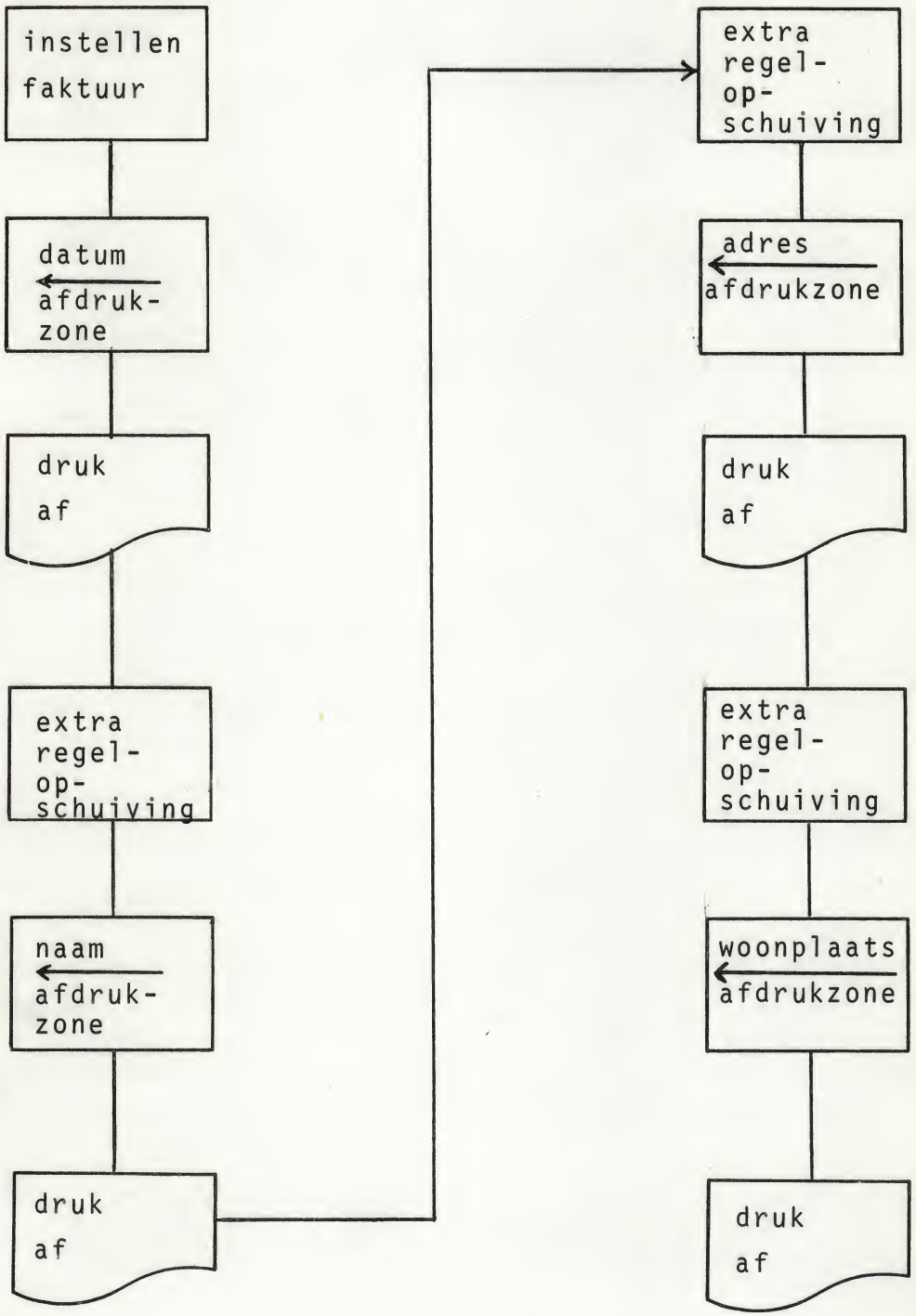
Voor het vervaardigen van stroomschema's wordt gebruik gemaakt van een template.

H. Wenteler

HOOFDBLOKSCHEMA

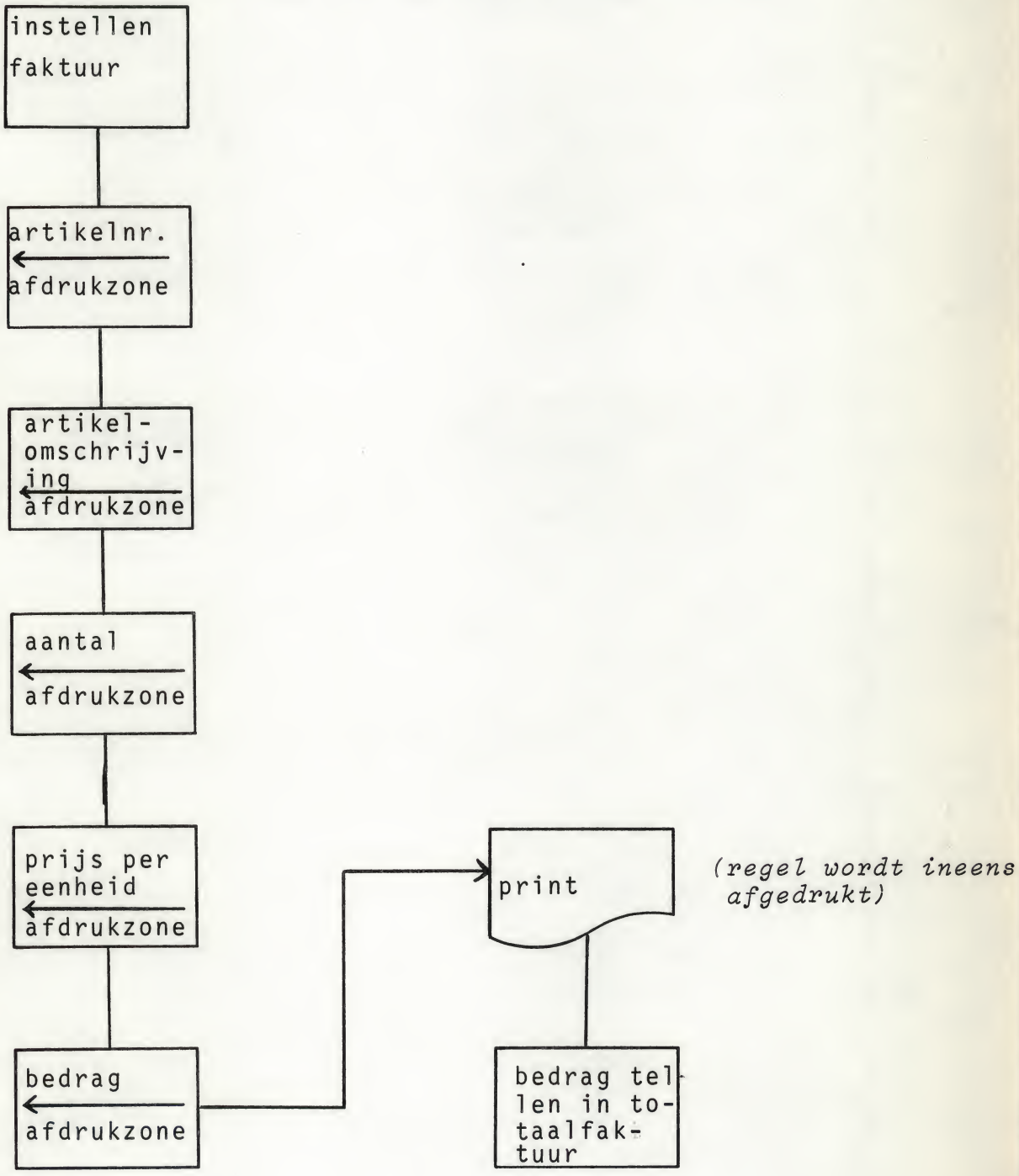


Specifikatie " faktuurkopregel"

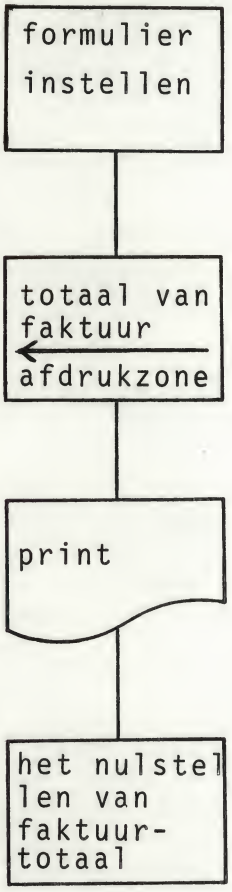


H. Wenteler

Specifikatie orderkaartprogramma

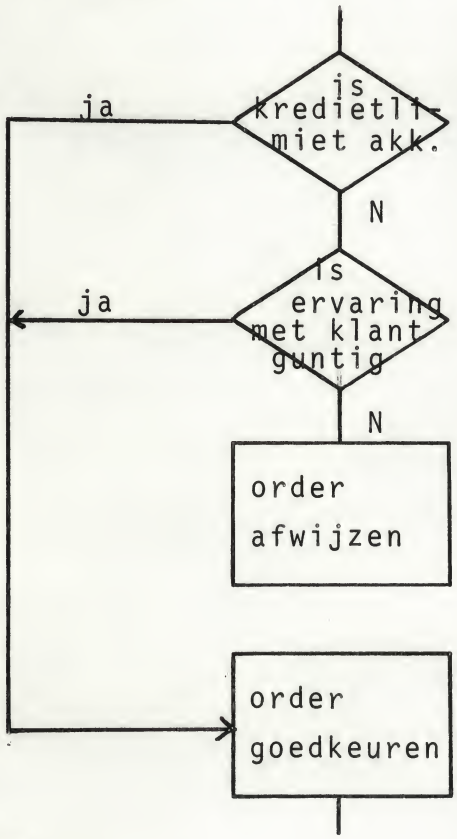


Specifikatie " afsluiten faktuur"

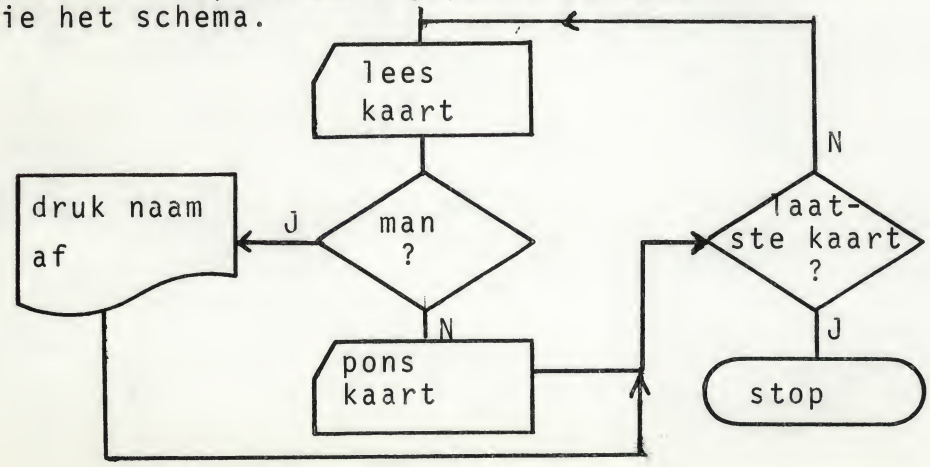


Voorbeeld 2
Uit een orderadministratie bekijken we het volgende onderdeel. Een order wordt geaksepteerd indien de kredietwaardigheid van de klant in orde is. Zo niet, dan wordt er bepaald of de ervaringen met de klant goed zijn. Is ook dat niet het geval, dan wordt de order afgewezen. In schemavorm kan dit er als volgt uitzien.

Detail :



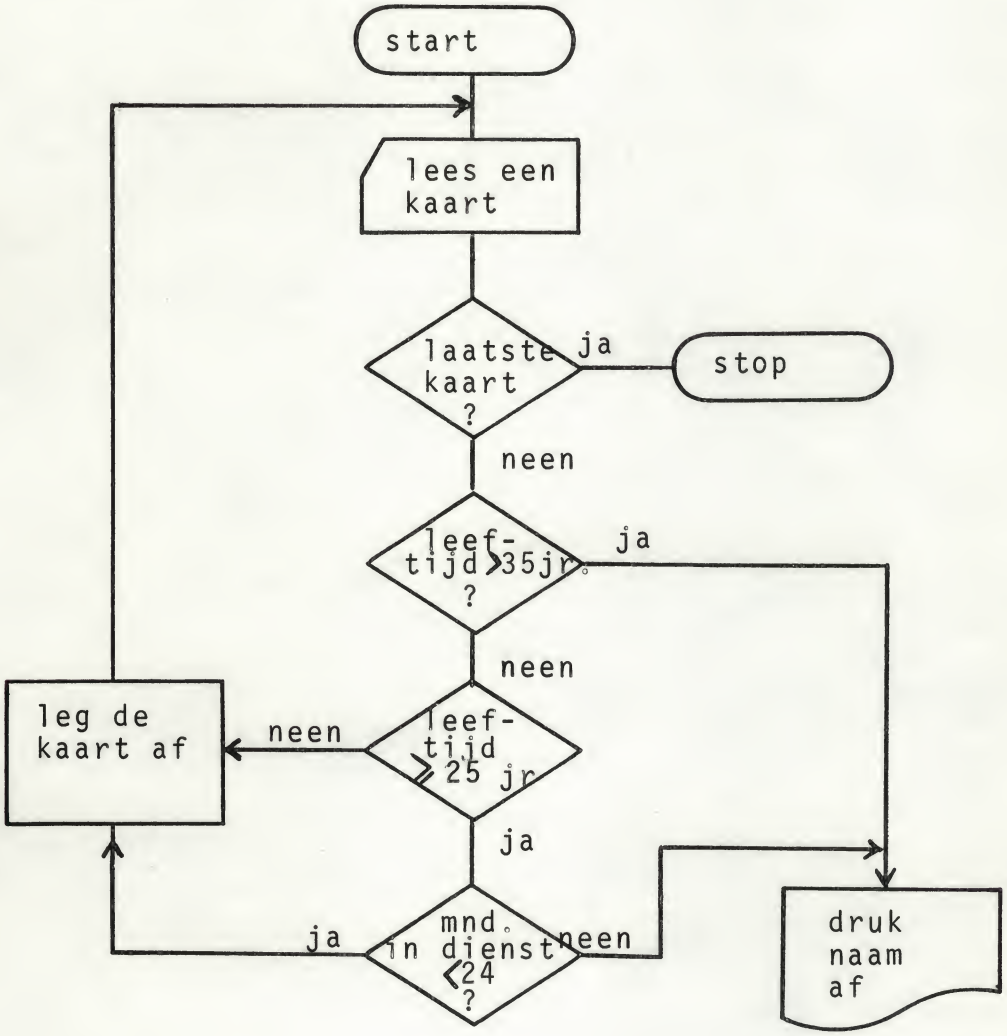
Voorbeeld 3
Een stapel ponskaarten uit de personeelsadministratie wordt door de machine gevoerd. Indien het een kaart van een mannelijke werknemer betreft moet de naam van de man afgedrukt worden via de sneldrukker, indien het een vrouw betreft moet er een ponskaart geponst worden.
Zie het schema.



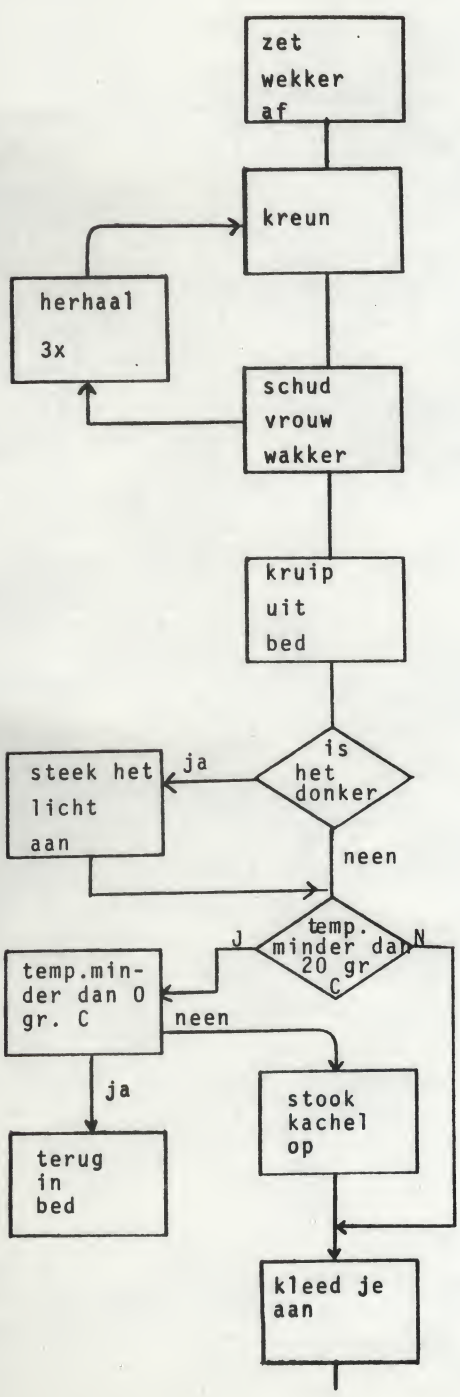
DE PRESENTATIE VAN EEN TE PROGRAMMEREN PROBLEEM

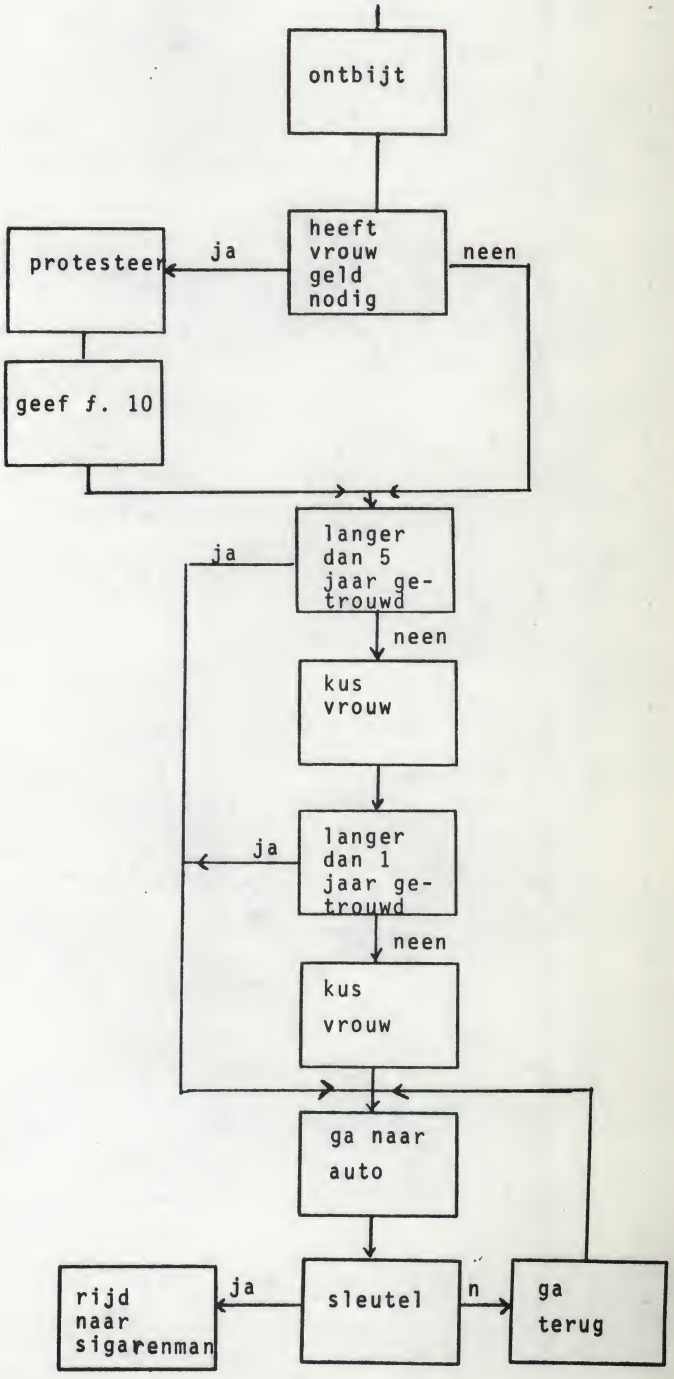
H. Wenteler

Beschikbaar is een pak kaarten met op elke kaart de informatie over een werknemer, n.l. de naam, leeftijd en aantal maanden in dienst.
De firma heeft besloten die personen in het pensioenfonds op te nemen, die ouder zijn dan 35 jaar of waarvan de leeftijd ligt tussen 25 en 35 jaar (dus t/m 35 jaar) en die minstens twee jaar in dienst zijn.
De programmeur moet nu de namen afdrukken van hen die in het pensioenfonds worden opgenomen.



Dat stroomschema's ook voor het beschrijven van andere problemen dan wetenschappelijke of administratieve, gebruikt kunnen worden laat bijgaand schema U duidelijk zien. Kommentaar lijkt ons hier overbodig.


zet wekker af
kreun
herhaal 3x
schud vrouw wakker
kruip uit bed
is het donker
ja
steek het licht aan
neen
temp. minder dan 20 gr C
J
N
temp.minder dan 0 gr. C
neen
ja
terug in bed
stook kachel op
kleed je aan


ontbijt
heeft vrouw geld nodig
ja
protesteer
neen
geef f. 10
langer dan 5 jaar getrouwd
ja
neen
kus vrouw
langer dan 1 jaar getrouwd
ja
neen
kus vrouw
ga naar auto
sleutel
ja
rijd naar sigarenman
n
ga terug

Ir. P.A. Tas

In dit hoofdstuk zullen voorbeelden behandeld worden die het nuttig gebruik van stroomschema's bij het schrijven van programma's duidelijk aantonen.

1. AFSPRAKEN

Allereerst zullen we echter enkele afspraken maken bij het gebruik van stroomschema's of zoals men ze ook wel noemt : Stroomdiagrammen.

1. *Symbolen worden door lijnen verbonden*
2. *De ingang van een symbool is van boven, de uitgang van onderen*
3. *Elk symbool, uitgezonderd de keuzehandeling, heeft slechts één ingang, en één uitgang.*
4. *Een lijn kan zich nooit splitsen zonder symbool, wel kunnen 2 lijnen samenkomen (meestal via een vaste connector)*
5. *Het tekenen van stroomdiagrammen van links naar rechts wordt veelvuldig toegepast.*

KORTE BESCHOUWING

De vorm van een stroomdiagram ligt niet vast. Men kan een beknopt, maar ook een zeer uitgebreid stroomdiagram van een probleem maken. Men kan een machine-afhankelijk of een machine-onafhankelijk stroomschema opzetten. Verkieslijk is een machine-onafhankelijk stroomdiagram. Hierbij worden geen speciale registers of geheugenplaatsen van een bepaalde machine genoemd maar wordt de tekst geheel symbolisch in de figuren geplaatst, zodat het mogelijk is met behulp van zo'n stroomschema een programma te schrijven in elke willekeurige code voor elke willekeurige machine.

Voor grote problemen is het zinvol een niet te gecompliceerd hoofdstroomschema te maken met een aantal bijbehorende detailstroomdiagrammen.

Er zijn programmeurs die menen, dat een stroomschema niet noodzakelijk is. Er moet echter de nadruk op gelegd worden dat de meeste programmeurs geen goed programma maken zonder een goed stroomschema. Bovendien is een stroomdiagram onontbeerlijk voor het vastleggen van het gecodeerde programma met bijbehorende informatie terwijl de overdracht van informatie betreffende de werking van het programma aanzienlijk vereenvoudigd wordt.

Door een uitgebreid stroomdiagram te maken kan het probleem goed geanalyseerd worden, waardoor fouten ontdekt en vermeden kunnen worden.

Hoewel het maken van een stroomdiagram met eventueel bijbe-

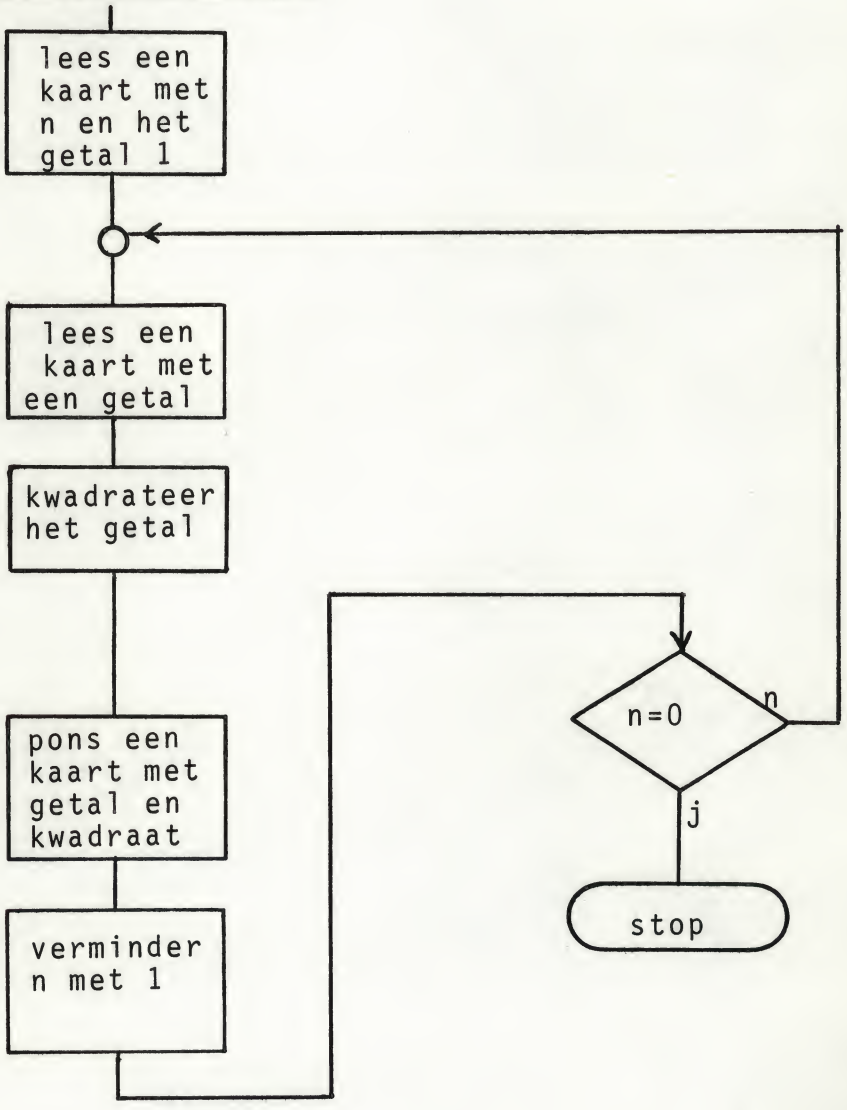
horende verklaring en het, tijdens het coderen, bijhouden van de veranderingen veel tijd kost, zal het uiteindelijke effect tijdwinst zijn. Dit wordt nog te weinig beseft.

3. PROGRAMMAVOORBEELDEN

Voorbeeld 1.

Gegeven : een aantal kaarten waarop in kolom 1 t/m 4 een getal van 4 cijfers. Deze kaarten worden voorafgegaan door een kaart die het getal n in kolom 1 t/m 4 bevat, dat aangeeft hoeveel kaarten verwerkt moeten worden, terwijl in kolom 5 het getal 1 staat.
Gevraagd : bereken het kwadraat van elk getal en pons dit in een kaart : in kolom 1 t/m 4 het getal
in kolom 8 t/m 15 het kwadraat.

Allereerst het stroomdiagram.



Ir. P.A. Tas

```
 0 | LSK          | Lees een kaart
 1 | SAB  1:4     | pos. 1-pos. 4 ⟶ B
 2 | KAG  4       | converteer van hexaden naar numeriek getal
 3 | BSB  22      | a ⟶ 22, + 0 ⟶ B
 4 | BPB  23      | + 0 ⟶ 23, + 0 ⟶ teller
 5 | KAB  5:5     | pos. 5 ⟶ B
 6 | KAG  1       | converteer van hexaden naar numeriek getal
 7 | BPB  24      | 1 ⟶ 24
 8 | LSK    ⟵     | lees een kaart
 9 | HAB  1:4     | pos. 1- pos. 4 ⟶ B
10 | KAG  4       | converteer
11 | BPB  25      | (B) = getal ⟶ 25
12 | VMG  25      | (B) x (25) ⟶ B
13 | KGA  8       | converteer naar hexaden (22) = n
14 | BAB  8:15    | breng naar buffer       (23) = teller
15 | PSK          | pons een kaart          (24) = 1
16 | HPA  23      | (23) ⟶ A                (25) = werkgeheugen
17 | OPA  24      | (23) + 1 ⟶ 23
18 | BPA  23      |
19 | AFA  22      | (A)-(22) =(teller) - n ⟶ A
20 | SNA  8  ⟶    | spring indien   < 0
21 | STP          | stop
```

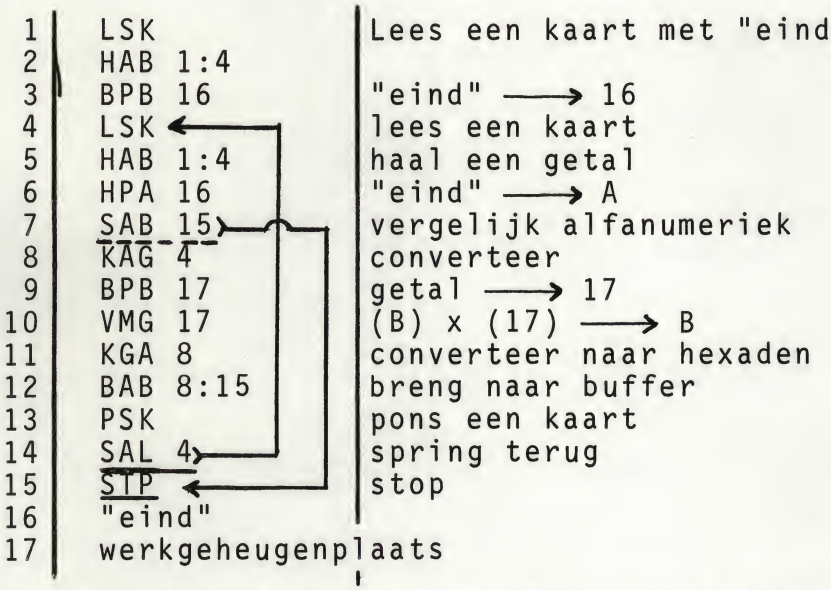
De instrukties 0 t/m 7 behoren tot de voorbereiding : het lezen van gegevens en teller op 0 brengen.
De instrukties 8 t/m 15 verrichten de opgave : een getal lezen, kwadrateren en uitponsen.
De instrukties 16 t/m 20 verhogen de teller en testen of het aangegeven aantal kaarten n verwerkt is. Is dat niet het geval dan wordt teruggesprongen naar adres 8 ; de conditie is hier dus : spring indien (A) $<$ 0.
Indien de (A) = 0 dan wordt de spronginstruktie genegeerd en de instruktie op adres 21 uitgevoerd die het programma stopt.
Drukken op de starttoets doet het programma weer aanvangen op adres 0. (Indien adres 0 bedoeld wordt, behoeft de 0 niet in het adresgedeelte geschreven te worden) Bij het lezen van een kaart worden de eerste 80 posities van de buffer gevuld. Bij de HAB- opdracht wordt de inhoud van de buffer niet verstoord.
De BAB 8:15 brengt het kwadraat van het getal naar de posities 8 t/m 15 en verstoort de andere posities niet zodat bij het ponsen van de kaart het getal in de kolommen 1-4 geponst wordt.

Voorbeeld 2

Bij voorbeeld 1 was bekend hoeveel kaarten verwerkt moesten worden. Stel het aantal onbekend en voeg een sluitkaart toe die in de kolom 1-4 het woord "eind" bevat.

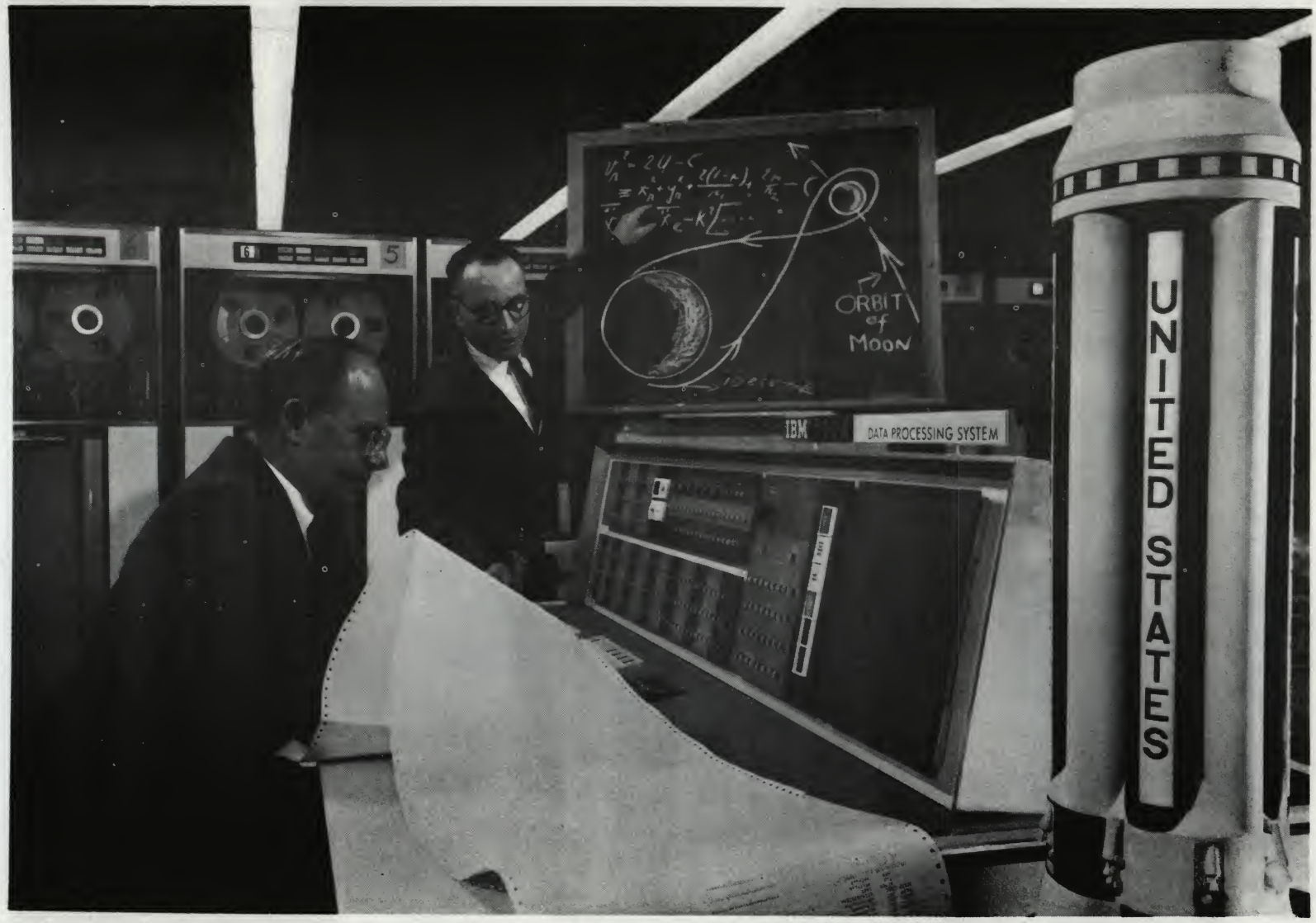
Ir. P.A. Tas

Het programma moet nu elke kaart na het inlezen testen op het woord "eind". Wordt dit gevonden dan betekent dit het einde van de verwerking en het programma kan gestopt worden.
De eerst in te lezen kaart bevat in kolom 1-4 ook het woord "eind", zodat deze informatie in het geheugen opgeslagen kan worden voor het vergelijken.



Voorbeeld 1 heeft enige interessante aspekten.
1. Een stuk van het programma wordt meerdere malen doorlopen.
2. Er vindt een telling plaats in een teller.
   Elke keer dat een bepaald stuk van het programma doorlopen is, wordt de teller met 1 (de telconstante) verhoogd.
3. Na elke verhoging van de inhoud van de teller wordt er een test uitgevoerd.

Deze konstruktie, het herhaaldelijk doorlopen van een stuk programma, wordt lus genoemd.
De lus, gecombineerd met telling en test, komt bijzonder vaak voor.



In de ruimtevaart neemt de computer een belangrijke plaats in.

IN- EN UITVOERAPPARATEN

Prof. Ir. D.H. Wolbers

Een publikatie van de Stichting Het Nederlands Studiecentrum voor Informatica, Amsterdam.

3. SYMBOLISCHE ADRESSEN

In een vorig hoofdstuk is reeds aangegeven, dat het invoerprogramma ons behulpzaam is om programma's eenvoudiger te kunnen schrijven. Het eerste voorbeeld daarvan was de invoering van een mnemotechnische code voor het operatiegedeelte van een instruktie. We zullen thans nagaan hoe het invoerprogramma ons kan helpen bij de adressering.
Daartoe beschouwen we nog eens de voorbeelden uit de vorige hoofdstukken.
Wat betreft het adresgedeelte van de instrukties kunnen we de instrukties verdelen in 2 groepen.
De eerste groep, waaronder haal-, breng- en rekeninstrukties, gebruikt het adres altijd als een plaats waar een getal staat of naar toegebracht moet worden.
Deze plaatsen noemen we wel de *WERKRUIMTE* van het programma.
De tweede groep, n.l. de spronginstrukties, hebben altijd een adres dat verwijst naar een plaats in het programma zelf.

De administratie, waar iedere grootheid in de werkruimte staat of moet staan, behoeven wij de machine niet mede te delen, want voor de machine volgt dit automatisch uit de instrukties. Tijdens het samenstellen van het programma moeten we dit echter wel voor ons zelf doen, opdat we bepaalde adressen niet meerdere keren gebruiken, terwijl we anderzijds ook een zo nuttig mogelijk gebruik willen en moeten maken van de beschikbare geheugenruimte. Vooral voor grotere problemen levert dit veel praktische bezwaren op met bovendien daardoor kans op vergissingen.De verwarring wordt daarbij in de hand gewerkt, omdat in de instrukties alleen voorkomen de adressen waar de gegevens staan. Deze adressen zelf hebben echter geen enkele relatie tot de getallen, die er in staan.

LABELS

Een geheel andere moeilijkheid geeft de tweede groep van instrukties. In alle gegeven voorbeelden zijn de instrukties genummerd aangegeven, in de vorm van een adres voor de kantlijn. Voor het invoerprogramma is dit niet nodig, want die kan dat door een telling zelf bijhouden. Bij het schrijven van het programma hebben we de nummering echter wel nodig om de adressen van de spronginstrukties te kunnen invullen. We nummeren nu alleen die plaatsen van het programma, waarnaar door spronginstrukties verwezen wordt.
Dergelijke nummers noemt men labels.

Aan het invoerprogramma laat men het over de correspondentie te bepalen tussen labelnummers en werkelijke adresnummers. Bij labels gebruikt men niet alleen getallen maar ook letters. Men kiest dan zinvolle benamingen, zoals loonberekening,

belastingtabel enz.
Ook combinaties van letters en cijfers is mogelijk zoals art. 568, enz. Volgens afspraak beperken we het aantal letters en cijfers tot 8. Bovendien eisen we dat zo'n naam altijd begint met een letter.

Voorafgaande voorbeelden moeten we dus beperken tot : loonber.,beltabel, art 568.

SYMBOLISCHE ADRESSEN

Minder dan 8 symbolen mag dus wel. Dergelijke labelnamen bestaande uit letters en cijfers noemt men symbolische adressen.

Een stukje programma kan er dan, wat betreft de sprongin-strukties als volgt uitzien :

```
PIET |
     |
     |
     |
     |   SAL PIET
     |
     |   SAL JAN
     |
     |
JAN  |
     |
     |   SPA JAN
     |
     |
     |
     |
```

Het ligt voor de hand om deze makkelijke methode nu ook te gebruiken voor de andere type instrukties.
Het programma voor berekening van x = a+ b - c - d luidt dan :

```
     |   HPA  A
     |   OPA  B
     |   AFA  C
     |   AFA  D
     |   BPA  X
```

DECLAREREN EN ASSEMBLER

Elders in het programma moeten we nu nog wel ergens voor de kantlijn de namen, A, B, C, D en X vermelden. Dit laatste noemen we het declareren van de symbolische adressen. Een invoerprogramma, dat bovengenoemde mogelijkheid biedt, noemen we assembler.

Ir. D.H. Wolbers

Een ander voorbeeld van symbolisch adresseren :

```
        HPB  BRUTLOON
        AFB  BELAST
        BPB  NETLOON
```

Deze methode van adresseren maakt het programmeren een stuk eenvoudiger. Men dient echter steeds voor ogen te houden : een *symbolisch adres* blijft een *adres*.

De in het laatste voorbeeld geschreven opdracht HPB BRUTLOON komt d.m.v. het invoerprogramma in het geheugen te staan als een HPB met een echt adres. De werking blijft ook haal de *inhoud* van dat adres naar B. We hebben aan dat adres echter een symbolische naam gegeven, die overeenkomt met de grootheid die daar in het geheugen staat. Daardoor wekt het geschreven programma de indruk alsof er staat *"haal het brutoloon in de B-accumulator"*.
Dit kan aanleiding geven tot fouten in die gevallen, waarbij we met de adressen zelf willen gaan rekenen. Door invoering van de symbolische adressen is de adresnummering komen te vervallen.

Doet zich het geval voor dat we vanaf verschillende plaatsen in een programma moeten verwijzen naar een aantal dicht bij elkaar gelegen andere adressen, dan moeten hiervoor toch allemaal verschillende namen gekozen worden.

```
          |
          |   SAL   PUNT 0
          |
          |   SAL   PUNT 1
          |
          |   SAL   PUNT 2
          |
          |   SAL   PUNT 3
          |
PUNT 0    |   AFA   KORTING 1
PUNT 1    |   OPA   TOESLAG 1
PUNT 2    |   AFA   KORTING 2
PUNT 3    |   OPA   TOESLAG 2
```

RELATIEF ADRESSEREN

Afhankelijk van de voorgeschiedenis van het programma moeten dus bepaalde kortingen en toeslagen met een gegeven bedrag in A verrekend worden. Men kan het declareren van de meerdere

namen vermijden door gebruik te maken van relatief adresseren. Deze addenden worden van de symbolische namen gescheiden door een + of een - teken. Dit teken heeft daarbij ook de gebruikelijke betekenis.
Het vorige voorbeeld kan dan geschreven worden als :

```
        :
        SAL     PUNT
        :
        SAL     PUNT +1
        :
        SAL     PUNT +2
        :
        SAL     PUNT +3
        :
PUNT    AFA     KORTING 1
        OPA     TOESLAG 1
        AFA     KORTING 2
        OPA     TOESLAG 2
```

Let wel op het verschil tussen PUNT 1 en PUNT +1. Want PUNT 1 is een complete naam, terwijl PUNT +1 betekent 1 plaats verder dan de naam PUNT.

Het zelfde voorbeeld zou met positieve en negatieve addenden ook kunnen luiden :

```
        :
        SAL   PUNT -2
        :
        SAL   PUNT -1
        :
        SAL   PUNT
        :
        SAL   PUNT +1
        :
        AFA   KORTING 1
        OPA   TOESLAG 1
PUNT    AFA   KORTING 2
        OPA   TOESLAG 2
```

Addenden moeten getallen zijn en mogen zelf geen symbolische adressen meer zijn. Niet toegelaten is dus een instruktie van de vorm

Ir. D.H. Wolbers

SAL JAN + PIET

3.1 INZETAANWIJZINGEN

We schrijven ons programma in mnemotechnische code en met symbolische adressen. Daarna wordt het programma geponst en door het invoerprogramma ingelezen. Daarbij moet echter worden aangegeven vanaf welke plaats.
Dit geven we dan ook aan door een 3-letter code met adres

```
BGN   n  begin in te zetten vanaf adres n.
```

Dit noemen we een *inzetaanwijzing*. Het heeft op papier de vorm van een instruktie, maar komt niet als zodanig in het geheugen. Het is in feite een "instruktie" ten behoeve van het invoerprogramma zelf.

Veelal zullen we een programma inzetten vanaf het begin van het geheugen.
We kunnen dan volstaan met

```
BGN
- - - -
- - - -
```

omdat ook hier een adres 0 mag worden weggelaten. Stel dat we willen beginnen vanaf adres 1000, dan begint ons programma met

```
BGN  1000
 ⋮
```

Deze inzetaanwijzing kunnen we ook gebruiken om ergens in ons programma een werkruimte te scheppen. Neem aan dat een programma moet werken met een serie van 100 getallen. Deze komen in opeenvolgende adressen.

Het programma werkt met deze getallen door middel van relatieve adressen ten opzichte van de symbolische naam LYST.
Het programma zelf begint gewoon op 0.
Dan is de volgende uitvoering mogelijk

```
        BGN
JAN                          } aantal instrukties
        SPA   PIET
        SAL   JAN
LYST    BGN   LYST + 100
PIET                         } overige instrukties
```

Achter de instruktie SAL JAN blijven nu automatisch 100 plaatsen in het geheugen vrij. Het geschreven programma is nu in 2 delen gesplitst. Tijdens de uitvoering van het programma zijn deze delen echter door spronginstrukties met elkaar verbonden. Het verschil tussen geschreven volgorde en die tijdens de uitvoering geven we vaak aan met woorden *STATISCH* en *DYNAMISCH*. De statische vorm is de vorm zoals het aan het invoerprogramma wordt aangeboden.
De dynamische vorm is de vorm zoals het programma door het besturingsorgaan van de machine later wordt uitgevoerd.

Met de besproken inzetaanwijzing BGN kunnen we zorgen, dat het invoerprogramma de opgegeven instrukties in het geheugen plaatst.
Wanneer alle instrukties ingelezen zijn moet dit ook aan het invoerprogramma worden medegedeeld.
Daartoe gebruiken we de volgende inzetaanwijzing

SRT n Start op adres n

Ook deze instruktie komt zelf niet in het geheugen. Het resultaat is echter, dat het invoerprogramma zijn funktie beeindigt en er voor zorgt dat als eerstvolgende instruktie door de machine wordt uitgevoerd de opdracht SAL n.
Als n kiezen we het adres van de eerst uit te voeren instruktie van het programma.

VOORBEELD VOOR EEN COMPLEET PROGRAMMA

Als voorbeeld voor een compleet programma nemen we het volgende probleem.
Gegeven 1000 ponskaarten, waarin in de kolommen 1 t/m 5 getallen staan.
Gevraagd 1000 nieuwe ponskaarten met dezelfde getallen in kolommen 1 t/m 5 en bovendien het kwadraat van de getallen in de kolommen 6 t/m 15. In beide gevallen de getallen zonder teken.

```
          | BGN
BEGINPRO  | STP   VOLGEND
VOLGEND   | BSA   TELLING
          | BSA   TELLING
BEGINCYC  | LSK
          | HAB   1:5
          | KAG   5
          | BPB   werk
```

Ir. D.H. Wolbers

```
                VMG     WERK
                KGA     s 10
                BAB     6:15
                PSK
                HPA     TELLING
                OPA     EEN
                BPA     TELLING
                AFA     DUIZEND
                SNA     BEGINCYC
                SAL     BEGINPRO
EEN             1
DUIZEND         1000
TELLING         0
WERK            0
                SRT     BEGINPRO
```

Aan het begin van het programma is een stopopdracht opgenomen. Na het inlezen van de programmakaarten heeft men dan gelegenheid de getalkaarten in te leggen. Op deze instruktie stopt de machine ook nadat de 1000 kaarten zijn verwerkt. Desgewenst kan men op deze manier meerdere series van 1000 kaarten verwerken.
Aan het eind van het programma zijn de nodige constanten en werkregisters aangegeven.
Voor het invoerprogramma *MOET* op iedere regel een instruktie of een getal worden gegeven. De nullen op de adressen TELLING en WERK mogen daarom niet worden weggelaten ; wel mag men er een ander getal of instruktie voor schrijven.

3.2 HET SPECIALE KENMERKKARAKTER

Wanneer een SERA woord als een instruktie wordt opgevat, komt daar één tetrade in voor, die de instruktie enkele bijzondere kenmerken kan geven. De betekenis daarvan kan nu worden aangegeven.
Zoals reeds uit voorbeelden is gebleken, heeft men in programma's vaak kleine constanten nodig.
Bij tellingen b.v. de telgrens en het getal, dat de toename aangeeft na iedere cyclus.
Deze getallen zijn meestal niet groter dan 1000 en nemen dus niet meer ruimte in dan het adresgedeelte van een instruktie.
Plaatsen we nu een dergelijk getal werkelijk als adres van een instruktie, dan dient deze natuurlijk anders te worden uitgevoerd.

Ir. D.H. Wolbers

Wanneer we een instruktie op deze wijze wensen te laten uitvoeren geven we dit op papier aan door tussen operatie en adres een ster te plaatsen.

Nemen we als voorbeeld de opdracht HPA

HPA n betekent (n) ⟶ A
HPA n betekent n ⟶ A

Dus in plaats van de inhoud van n gaat nu het getal n zelf naar A. Het invoerprogramma verwerkt deze ster en zorgt dat de kenmerktetrade 0001 wordt.
Deze mogelijkheid geldt voor alle haal- en rekenopdrachten. Bij de spronginstrukties heeft de stermogelijkheid geen betekenis, omdat hier het adres reeds direkt de volgende uit te voeren instruktie aangeeft.

Een voorbeeld van een telling opklimmend met 1 en als grens 100 kan dan als volgt geprogrammeerd worden:

```
HPA   TELLING
OPA * 1
BPA   TELLING
AFA * 100
SOA
```

De sterfaciliteit mag ook gebruikt worden wanneer het adres als symbolische naam gegeven is. Wanneer b.v. de naam PIET in werkelijkheid behoort bij adres 365, dan betekent HPA PIET haal de inhoud van 365 naar A, maar HPA PIET heeft tot gevolg dat het getal 365 naar A gaat. Deze toepassing kan van belang zijn bij nog te bespreken subprogrammatechniek.

DOOD GEHEUGEN

Een ander gebruik van het kenmerkkarakter is het volgende. Tot nu toe is aangenomen, dat het geheugen van de SERA bestaat uit 10000 plaatsen, waarin we zowel kunnen schrijven als lezen. Nu wordt echter het invoerprogramma geplaatst in een extra stuk geheugen waarin we wel kunnen lezen maar niet schrijven.
Zo'n geheugen heet dood geheugen. Op technische manier wordt daar de eerste keer de gewenste informatie ingebracht. Het voordeel is dat nu niet door een eigengemaakt programma met fouten dit invoerprogramma verstoord kan worden.

Ook dit dode geheugen is weer genummerd van 0 t/m 9999 en als decimaal getal rechts in een instruktie aangegeven.
Voor onderscheid tussen levend en dood geheugen dient nu het 2e bit van links van de kenmerktetrade.

Heeft het adres betrekking op het levend geheugen dan is de kenmerktetrade 0000, terwijl in het geval van het dode geheugen dit 0100 wordt. Een aantal plaatsen in het dode geheugen zijn gemarkeerd door vaste symbolische namen.Gebruikt men deze namen in een programma, dan verzorgt het invoerprogramma op de juiste manier het kenmerk.
Voor alle andere namen alsmede echte adressen wordt automatisch aangenomen dat deze betrekking hebben op het levend geheugen. Mocht men abuisievelijk een dergelijke vaste naam toch gebruiken voor eigen doeleinden, dan zal men dus ook een eigen declaratie geven. In dat geval "vergeet" het invoerprogramma tijdelijk de vaste naam en verzorgt dan toch een juiste adressering.

HET ADRESSEREN VAN SPECIALE REGISTERS

Tenslotte bestaat er vaak behoefte om speciale registers ook te kunnen adresseren. Voorbeelden daarvan zijn de A- en B-accumulator.
Dit soort registers behoort noch tot het levend noch tot het dode geheugen. Desondanks zijn ook deze registers weer genummerd vanaf 0 en worden als normaal adres aangegeven. In dat geval wordt het kenmerk echter 0010.
Ook aan deze registers zijn vaste namen gegeven, die zonder declaratie mogen worden gebruikt. Tevens geldt ook hier weer dat een eigen declaratie de betekenis van de vaste naam buiten gebruik stelt.Voor A- en B-accumulatoren gelden de namen ACCU-A en ACCU-B.
Toepassingen van deze namen zijn bijvoorbeeld :

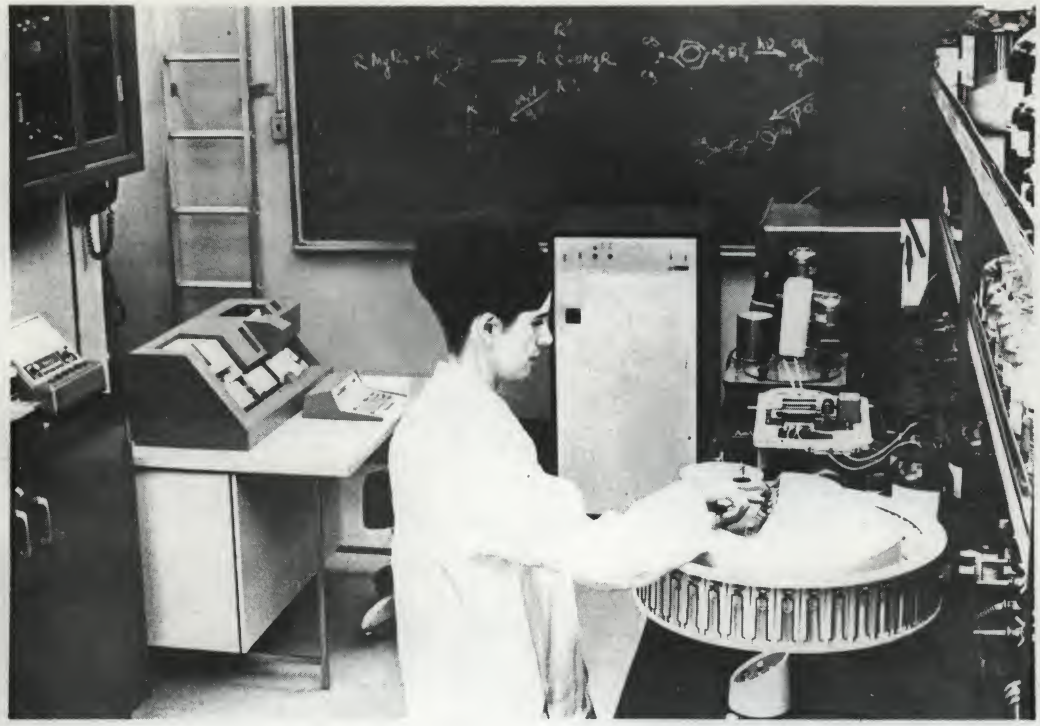
```
HNA   ACCU-A        verwissel het getal in A van teken
HPA   ACCU-B        breng de inhoud van B naar A
VMG   ACCU-B        vorm het kwadraat van getal in B.
```

Samenvattend geldt voor het kenmerkkarakter :

in decimale vorm	in binaire vorm	adresgedeelte wordt gebruikt
0	0000	voor levend geheugen
1	0001	als direkt getal
2	0010	voor speciale registers
4	0100	voor dood geheugen

3.3 SUBPROGRAMMA'S

Bij het samenstellen van een groot programma komt het vaak



bloedonderzoek

COMPUTER TOEPASSINGEN

in de sport

voor dat op verschillende plaatsen in het programma dezelfde series instrukties gebruikt moeten worden. Deze identieke series vertegenwoordigen daarbij ook inderdaad dezelfde soort berekening of verwerking van gegevens, maar kunnen door de overige eisen in het programma niet tot een lusprogramma verenigd worden.
Vooral wanneer een dergelijke serie instrukties betrekkelijk lang is, is het duidelijk dat het meermalen voorkomen een zekere verspilling van geheugenruimte is.
Daarnaast maakt het ook het programmeren nodeloos moeilijker, omdat zonder meer copiëren voor de volgende series vanaf de eerste ook niet mogelijk is vanwege de andere adresplaatsen.

We kunnen de genoemde bezwaren ondervangen door een dergelijke serie instrukties slechts één maal ergens in het geheugen te plaatsen. Vanaf iedere positie in het grote programma, waar we deze serie willen gebruiken kunnen we dan daar naartoe springen, echter met dien verstande, dat na het doorlopen van de serie weer teruggesprongen wordt naar het punt vanwaar deze serie als het ware werd opgeroepen.

Dit betekent dat bij de sprong naar de gewenste serie instrukties ook de informatie vanaf welke plaats deze sprong plaats vond, meegenomen moet worden.
Hoewel het in principe wel mogelijk is met de tot nu toe besproken instrukties een dergelijke programmering te verwezenlijken, komt deze werkwijze zo vaak voor, dat men in iedere rekenmachine daarvoor een speciaal mechanisme in de vorm van één of meerdere speciale instrukties heeft ingebouwd.

Een serie zelfstandige instrukties voor meermalig gebruik noemt men dan een *SUBPROGRAMMA* of soms ook wel *subroutine*.

Ter onderscheid spreekt men dan ook vaak over *HOOFDPROGRAMMA* als programma dat de oproepende spronginstrukties bevat.

BIBLIOTHEEK VAN SUBPROGRAMMA'S

Door het min of meer zelfstandige karakter van een subprogramma, kan men dit alleen gebruiken t.b.v. het hoofdprogramma, waarvoor het oorspronkelijk gemaakt is, maar voor ieder ander hoofdprogramma dat dezelfde deelbewerking vereist. Dit is zelfs de oorzaak dat voor iedere machine een meer of minder omvattende bibliotheek van subprogramma's bestaat, die reeds bij voorbaat gemaakt zijn voor de meest voorkomende procédés, zelfs nog voor er van concrete toepassing sprake is.
Hierbij b.v. te denken aan subprogramma's voor het sorteren van gegevens al of niet met magneetbanden, de uitvoer via sneldrukker met een voorgeschreven lay out, voor wetenschappelijke toepassingen subprogramma's voor het berekenen van

standaardfunkties zoals log, sin, enz.

Naarmate men verwacht dat zo'n subprogramma vaker gebruikt zal worden kan men er meer of minder aandacht aan schenken bij het samenstellen.
Afhankelijk van de toepassingen zal men ook de nadruk leggen op een dynamisch snelwerkend programma of een programma, dat statisch zo klein mogelijk is. In de meeste gevallen zijn dit n.l. tegenstrijdige eisen.
In SERA wordt het oproepen van subprogramma's gerealiseerd door een speciale sprongopdracht, de zgn. subprogrammasprong.

SSP n Spring naar subprogramma dat begint op adres n en plaats de terugkeerinstruktie in het register TERUG

Ter verklaring van de opdracht het volgende.
Er is een speciaal register aanwezig ter grootte van een volledig SERA woord, dus 48 bits. Dit register is programmatisch, waar nodig, bereikbaar onder de vast gekozen symbolische naam TERUG.

In tegenstelling tot alle instrukties, die we tot nu toe leerden kennen is voor de SSP instruktie van belang het geheugenadres, waar deze SSP instruktie staat. Nemen we aan dat dit is adres P. De werking van de SSP opdracht omvat dan 2 handelingen die achtereenvolgens worden uitgevoerd.

1e. In het speciale register TERUG wordt een absolute sprong-instruktie met adresdeel p + 1 geplaatst, dus de inhoud van TERUG wordt SAL p + 1.
2e. Er vindt een absolute sprong naar adres n plaats, d.w.z. het is alsof nu uitgevoerd wordt de instruktie SAL n.

Na uitvoering van een SSP instruktie komen dus automatisch alle instrukties van het aangeroepen subprogramma aan de beurt. Deze instrukties hangen volkomen af van het gewenste procédé, dat tijdens de werking van het subprogramma, moet worden uitgevoerd. De enige uitzondering hierop is de laatste instruktie, die moet dienen om weer naar het hoofdprogramma terug te keren. Deze laatste instruktie luidt nu echter SAL TERUG. De uitvoering van deze instruktie heeft tot gevolg een absolute sprong naar het speciale register TERUG. Maar daar staat, tengevolge van de vroeger uitgevoerde SSP instruktie, SAL p + 1, waardoor automatisch een terugsprong uitgevoerd wordt naar het punt direkt volgend op de genoemde SSP opdracht.

Tijdens de werking van een subprogramma is het vrijwel altijd noodzakelijk, dat gerekend wordt met één of meerdere gegevens uit het hoofdprogramma, die overigens bij iedere

oproep verschillend kunnen zijn. Omgekeerd levert een subprogramma ook meestal één of meerdere resultaten t.b.v.het hoofdprogramma.
Tijdens de aanroep en de terugkeer moeten dus gegevens resp. resultaten "meegenomen" worden.
Betreft dit slechts 1 of 2 grootheden, dan neemt men hiervoor bij voorkeur een of beide der accumulatoren.
Betreft het meerdere grootheden dan zijn er een aantal mogelijkheden, die echter in het volgend hoofdstuk besproken zullen worden.

VOORBEELD VAN EEN SUBPROGRAMMA

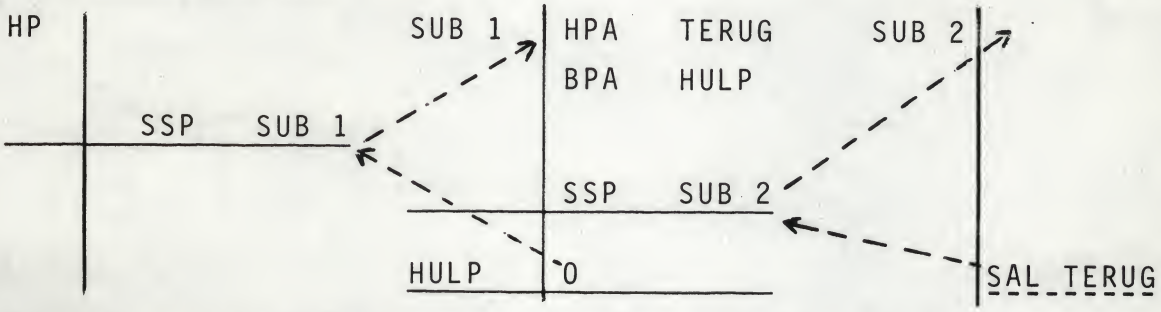
Als voorbeeld van een subprogramma nemen we een zeer simpel rekenproces.
Gegeven zijn 2 getallen A en B. Na terugkeer uit het subprogramma moet in A staan het verschil en in B de som van de getallen.
Het geval van overloop wordt buiten beschouwing gelaten.
Speciaal ter verduidelijking wordt hier nog gebruik gemaakt van absolute adressen.Vanaf 4 plaatsen in het hoofdprogramma t.w. 100, 133, 500 en 750 wordt het subprogramma op adres 1000 ingeroepen.

```
 100 |  SSP   1000
 133 |  SSP   1000
 500 |  SSP   1000
 750 |  SSP   1000
1000 |  AFA   ACCU B
1001 |  OPB   ACCU B
1002 |  OPB   ACCU A
1003 |  SAL   TERUG
```

Tijdens de dynamische afloop van het programma gebeurt dan het volgende. Aangekomen bij de eerste SSP opdracht op 100 wordt de instruktie SAL 101 in TERUG geplaatst. Dan vindt een sprong naar 1000 plaats.
De instrukties op 1000 t/m 1002 zorgen nu voor het uitvoeren van de gewenste berekeningen en daarna vindt een sprong plaats naar TERUG, waarna via de sprong SAL 101 teruggesprongen wordt in het hoofdprogramma. Het totale resultaat is dus dat het geven van de SSP opdracht het eenmalig aflopen van het gehele subprogramma veroorzaakt, waarna het hoofdprogramma normaal zijn weg vervolgt.
Hetzelfde vindt plaats bij de volgende oproepen vanaf plaatsen 133, 500 en 750 waarbij dan sprongopdrachten naar resp. 134, 501 en 751 in TERUG geplaatst worden. Een subprogramma

Ir. D.H. Wolbers

kan op zijn beurt een ander subprogramma oproepen. In SERA moet men in dat geval programmatisch enige maatregelen nemen omdat slechts één register TERUG beschikbaar is.
Deze maatregelen komen hierop neer dat men in het eerste subprogramma alvorens de sprong naar het tweede subprogramma uit te voeren, de inhoud van TERUG in veiligheid brengt.
Dit kan als volgt :



Door de eerste twee instrukties van SUB 1 wordt de "terugkeerinstruktie" geplaatst op de laatste adresplaats van SUB 1, hier aangegeven met de naam HULP.
Zou SUB 2 op zijn beurt weer een derde subprogramma moeten aanroepen dan moet men in SUB 2 opnieuw een dergelijke serie instrukties invullen. In sommige machines zijn daarom voor dit doel zelfs een aantal registers "TERUG" ingebouwd.
Bij de eerste oproep wordt automatisch de eerste genomen, bij een herhaalde oproep de tweede enz. Na iedere terugkeer wordt de telling als het ware weer teruggezet. Daarbij mag natuurlijk de "diepte" waarmee subprogramma's in elkaar genesteld worden natuurlijk het aantal registers "TERUG" niet overschrijden, want dan ontstaan dezelfde moeilijkheden als hierboven geschetst voor SERA bij de tweede oproep. Een ander aspekt van de terugkeerinstruktie is nog het volgende. Zoals bij bespreking, over het transport van gegevens en resultaten naar en van het subprogramma, nog zal blijken, heeft men daarbij de behoefte aan een gemeenschappelijk referentiepunt in het geheugen zowel voor het hoofdprogramma als voor het subprogramma.
Een voor de hand liggend punt is daarbij de plaats van de SSP sprong of het eerst volgend adres. Dit laatste is precies het adresgedeelte van de terugkeerinstruktie. Zou men dus in het subprogramma van dit adresgedeelte afzonderlijk gebruik willen maken, dan moet men dus als het ware het operatiegedeelte, in dit geval SAL, uit de terugkeerinstruktie verwijderen. Dit is programmatisch zeer wel mogelijk, omdat we instrukties ook als getallen kunnen opvatten en er dus mee kunnen rekenen.
Met de reeds besproken SERA opdrachten zouden we b.v.in het rekenorgaan een "instruktie" kunnen delen door het getal 10.000.

De rest bij deze deling geeft dan precies het adresgedeelte. Het gewenste resultaat is overigens met nog later te bespreken logische opdrachten nog eenvoudiger te bereiken. Het hiervoor genoemde gebruik van het "terugkeer adres" is echter aanleiding, dat in sommige rekenmachines bij de subprogrammasprong in het overeenkomstige register TERUG niet de terugkeerinstruktie maar alleen het terugkeeradres opgeborgen wordt.
Bij de werkelijke terugkeer naar het hoofdprogramma moet dan natuurlijk nog op de een of andere wijze de sprongoperatie worden toegevoegd aan dit terugkeeradres.

In enkele machines, waaronder ook SERA, zijn beide mogelijkheden verenigd en wel als volgt. De operatie SAL wordt in de machine vertegenwoordigd door de cijfercombinatie 00. Dit betekent dat in SERA een positief getal n van max. 4 cijfers opgevat als instruktie de betekenis heeft van SAL n. In SERA vallen daardoor de begrippen terugkeerinstruktie en terugkeeradres samen.

HET GEBRUIK VAN SYMBOLISCHE ADRESSEN IN EEN SUBPROGRAMMA

Tot slot bespreken we nog het probleem van het gebruik van symbolische adressen in een subprogramma. Wat betreft symbolische adressen vormt een subprogramma een geheel met het hoofdprogramma. Dit betekent dat namen, die in een subprogramma gebruikt zijn, niet meer gebruikt mogen worden in het hoofdprogramma en omgekeerd.
Hetzelfde geldt trouwens ook voor de namen in verschillende subprogramma's, die alle behoren bij het zelfde hoofdprogramma.
Dit betekent een ernstige beperking omdat vaak verschillende subprogramma's door verschillende mensen gemaakt worden. Men moet dan van te voren afspraken gaan maken omtrent de naamgeving.

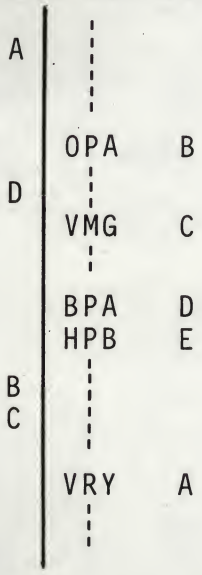
Nog moeilijker ligt de zaak voor algemene subprogramma's, die de ene keer in dit programma, en volgende keer in dat programma gebruikt worden.
Door het toch min of meer zelfstandig karakter van subprogramma's zullen de in een subprogramma gebruikte namen met de daar gegeven betekenis in de meeste gevallen alleen binnen dit subprogramma van belang zijn.

Een verwijzing van buiten het subprogramma naar namen binnen het subprogramma komt heel weinig voor. In verband met voorgaande ontstaat daardoor vanzelf de behoefte om de namen in een subprogramma een beperkte betekenis te geven, we spreken in dat geval van locale betekenis.
Buiten het subprogramma willen we deze namen dan weer "ver-

geten". De enige uitzondering is natuurlijk de naam, die we aan de eerste instruktie van een subprogramma geven en waarmee we het subprogramma d.m.v. een SSP opdracht kunnen aanroepen.
In sommige programmeersystemen maakt men daarom wel onderscheid tussen locale en globale namen. De locale namen hebben daarbij alleen betekenis in een speciaal aangegeven deel van het programma, de globale namen behouden hun geldigheid in alle delen.

In SERA is dit probleem opgelost door een speciale inzetaanwijzing VRY waardoor automatisch alle gebruikte namen in een aangegeven gebied locaal worden voor dit gebied en daarbuiten hun betekenis hebben verloren.
Bij voorkeur maken we een subprogramma tot een dergelijk gebied, waarmee we het beoogde doel hebben bereikt.

Het gebruik van de inzetaanwijzing volgt het eenvoudigst uit een voorbeeld.



Door de inzetaanwijzing VRY met als "adresdeel" een symbolische naam wordt het stuk programma gelegen tussen de plaats waar deze naam gedeclareerd is en deze inzetaanwijzing tot een bovengenoemd gebied verklaard.

BLOK

Een dergelijk stuk programma noemen we ook wel een blok. Alle namen die in een blok gedeclareerd zijn, zijn daardoor locaal voor dit blok. In het voorbeeld is precies een blok gegeven. In dit blok zijn B, C en D locale namen en hebben buiten dit blok de hier aangegeven betekenis verloren. Dit betekent tevens dat dezelfde namen daar dus opnieuw gedecla-

reerd mogen worden en daar dan onder hun nieuwe betekenis bekend zijn.
In het voorbeeld is E niet gedeclareerd in het blok. Deze declaratie moet ergens buiten het blok plaats vinden. De naam E is dus geen locale naam. In dit geval is ook de naam A niet locaal want door VRY A zijn alleen de namen tussen A en VRY A "vrijgemaakt". Deze conventie past dus ook mooi bij gebruik van subprogramma's.
Wanneer we ieder subprogramma afsluiten met VRY en dan daarachter de naam van het subprogramma, dan is buiten het subprogramma alleen zijn eigen naam bekend.

Nemen we nog eens het voorbeeld van de som en verschil van 2 getallen in A en B. We programmeren het nog iets anders waardoor we ook een inwendige geheugenplaats X nodig hebben.

```
MINPLUS | BPA  X
        | AFA  ACCU B
        | OPB  X
        | SAL  TERUG
      X | 0
        | VRY  MINPLUS
```

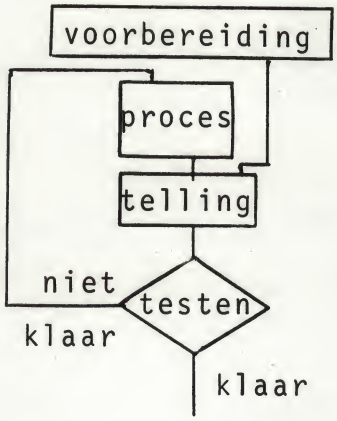
Buiten dit programma is dus alleen de naam MINPLUS bekend, terwijl de naam X is vergeten.

3.4 TELLINGEN

Reeds enkele malen hebben we voorbeelden ontmoet, waarbij een bepaald proces een aantal keren uitgevoerd moest worden. Bij zulke tellingen kunnen we vrijwel altijd 4 delen onderscheiden, n.l. voorbereiding, proces, telling en testen. De drie laatste delen zijn altijd cyclisch met elkaar verbonden hetgeen op 2 manieren kan plaatsvinden. Vanuit de voorbereiding kan men dan op 3 verschillende plaatsen in deze cyclus springen, hetgeen dan de 6 volgende mogelijkheden geeft :

1

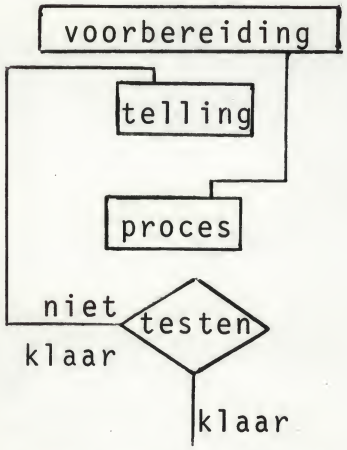
2



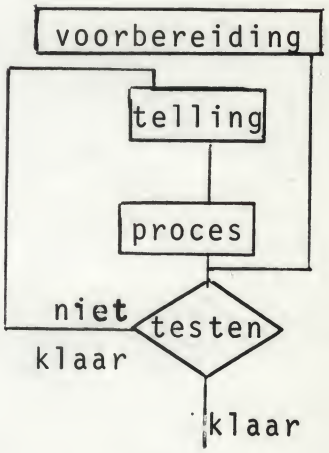
3

4

5



6



Voorkeur voor een bepaalde methode kan verschillende oorzaken hebben zoals aansluiting aan overige programma-onderdelen, al of niet gebruik van de telling binnen het proces, de waarde van de telling bij bereiken van de telgrens, het eventueel bij voorbaat nul zijn van de telgrens.
We geven een voorbeeld, dat alleen volgens het eerste en laatste schema wordt uitgewerkt.
Gevraagd wordt een subprogramma, dat een aantal kaarten leest. Iedere kaart bevat gegevens, maar de kolommen 1 t/m 3 zijn niet ingevuld. Hierin moet een volgnummer komen, te beginnen bij 1. Bij binnenkomst in het subprogramma staat het aantal kaarten in B.

```
VOLGNR | SOB  TERUG
       | OPB*1
       | BPB  TELGR
```

```
          HPB * 1
          BPB   TELLING
PROCES    LSK
          HPH   TELLING
          KGA   s3
          BAB   1:3
          PSK
          HPB   TELLING
          OPB * 1
          BPB   TELLING
          AFB   TELGR
          SNB   PROCES
          SAL   TERUG
TELLING   0
TELGR     0
          VRY   VOLGNR
```

Volgens het laatste schema wordt het zelfde programma als volgt :

```
VOLGNR    BSB   TELGR
          BPB   TELLING
          SAL   TEST
TEL       HPB   TELLING
          OPB * 1
          BPB   TELLING
          LSK
          KGA   s3
          BAB   1:3
          PSK
          HPB   TELLING
TEST      AFB   TELGR
          SNB   TEL
          SAL   TERUG
TELLING   0
TELGR     0
          VRY   VOLGNR
```

In het laatste geval behoeft dus niet tijdens de voorbereiding apart getest te worden op geval 0. Dit is in principe alleen mogelijk met de 2e, 3e en 6e methode. In de andere gevallen wordt het proces tenminste éénmaal uitgevoerd.

Bij lusprogramma's waarbij het aantal keren door een telling wordt bijgehouden komt het veelvuldig voor dat het proces zelf moet worden uitgevoerd op een serie gegevens, die ergens achter elkaar in het geheugen staan. Dit betekent dat de benodigde haal- en brengopdrachten voor deze gegevens en resultaten bij iedere doorgang van de lus een ander adresdeel moeten hebben. Dit wil dus zeggen dat het adresdeel van bepaalde opdrachten, tijdens de werking van het programma steeds moet veranderen.
Men moet dan eigenlijk bij iedere keer het adres "uitrekenen".

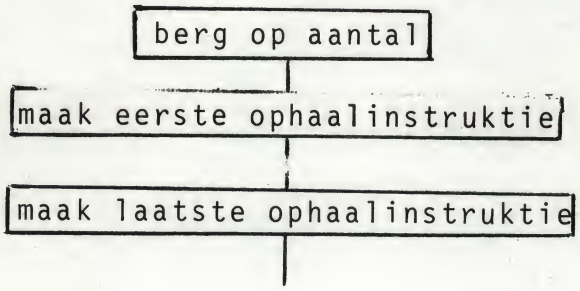
Zoals we reeds bij de bespreking van de terugkeerinstruktie gezien hebben is dat geen bezwaar.

VOORBEELDEN

Hierna zijn daarom nog 2 voorbeelden gegeven waarbij dit principe is toegepast. In een volgend hoofdstuk ontmoeten we nog een opdracht die deze "adresrekening" effectiever kan uitvoeren.

Het eerste programma berekent het gemiddelde van een serie getallen, die achter elkaar in het geheugen staan.
Bij binnenkomst in het subprogramma is gegeven in A het beginadres van de lijst getallen en in B het aantal getallen.
Bij terugkeer staat het gemiddelde in B als geheel getal.

Het tweede programma zoekt in een gegeven lijst getallen het kleinste en grootste exemplaar.
Ook hier geldt bij binnenkomst is het beginadres van de lijst in A en het aantal in B.
Bij terugkeer staat het kleinste in A en het grootste in B.



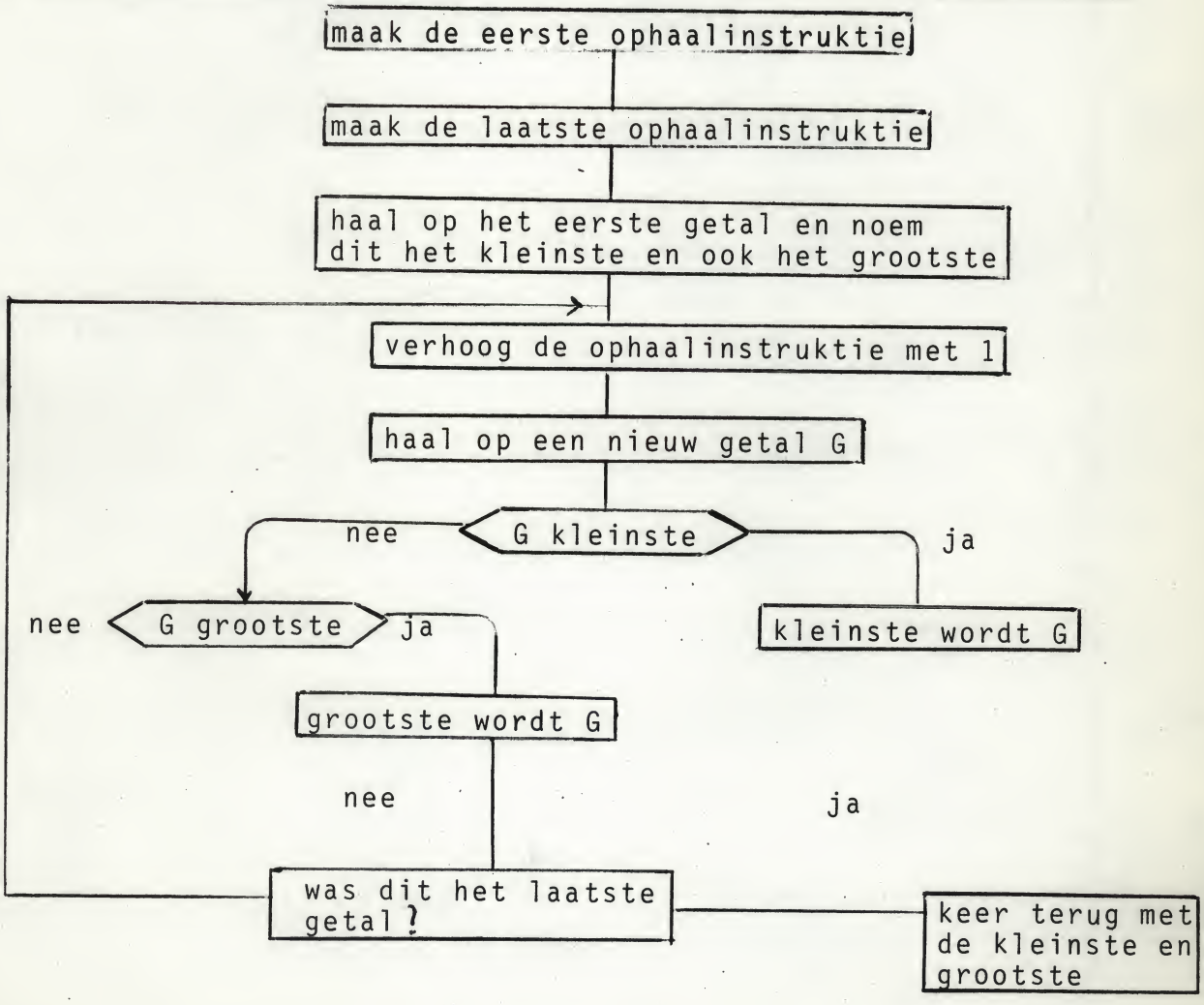

```
GEMID    BPB   AANTAL
         OPA   HULP
         BPA   VOLGEND
         OPA   ACCU B
         AFA * 1
         BSA   LAATSTE
         BPA   SOMKOP
         BSA   SOMST
VOLGEND  0
         OAB   SOMST
         OPA   SOMKOP
         BPB   SOMST
         BPA   SOMKOP
         HPA   VOLGEND
         HPB   LAATSTE
         SAB   KLAAR

         OPA * 1
         BSA   VOLGEND
         SAL   VOLGEND
```

Ir. D.H. Wolbers

```
KLAAR   | HPA   SOMKOP
        | HPB   SOMST
        | DLN   AANTAL
        | SAL TERUG
AANTAL  | 0
HULP    | HPB   0
LAATSTE | 0
SOMKOP  | 0
SOMST   | 0
        | VRY   GEMID
```

SUBPROGRAMMA VOOR BEPALEN VAN DE GROOTSTE EN KLEINSTE UIT EEN EEN GEGEVEN SERIE GETALLEN



```
KGLR    OPA   HULP
        BPA   EERSTE
        BPA   VOLGENS
        OPA   ACCU B
        AFA * 1
        BPA   LAATST
EERSTE  0
        BPA   KLEIN
        BPA   GROOT
OPN     HPA   VOLGEND
        OPA * 1
        BPA   VOLGEND
VOLGEND 0
        BPA   G
        AFA   KLEIN
        SNA   WELKL
        HPA   G
        AFA   GROOT
        SNA   NIETGR
        HPA   G
        BPA   GROOT
NIETGR  HPA   VOLGEND
        AFA   LAATST
        SOA   KLAAR
        SAL   OPN
WELKL   HPA   G
        BPA   KLEIN
        SAL   NIETGR
KLAAR   HPA   KLEIN
        HPB   GROOT
        SAL   TERUG
HULP    HPA   0
LAATST  0
KLEIN   0
```

samenstellen weerberichten

COMPUTER TOEPASSINGEN

plaatsreserveringen voor vliegtuigen

in ziekenhuizen

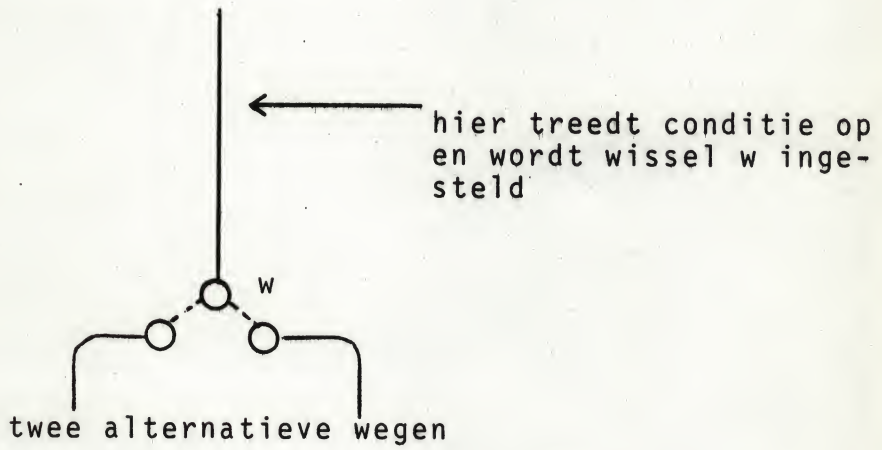
Ir. D.H. Wolbers

GROOT	0
G	0
	VRY KLGR

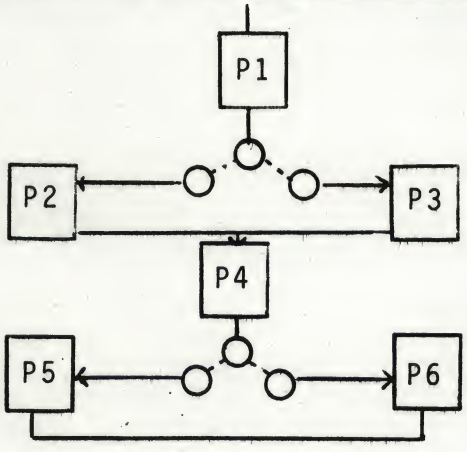
3.5. WISSELS

Het komt in een programma vaak voor, dat verschillende alternatieve delen moeten worden doorlopen afhankelijk van een bepaalde conditie, die reeds eerder in het programma is opgetreden. Men moet dan daar ter plaatse een *wissel* instellen ten behoeve van het latere programmadeel.

Schematisch ziet dat er als volgt uit :



Een wissel kan ook herhaald voorkomen, bijv. :



Ir. D.H. Wolbers

Dit stuk kan betrekking hebben op een loonberekening waar P1 en P4 altijd gelijke programmadelen zijn, terwijl P2 en P5, resp. P3 en P6 gelden al naar gelang de grens voor de sociale wetten wordt overschreden of niet.
Hoewel w1 en w2 als twee wissels zijn aangegeven, zijn ze van dezelfde conditie afhankelijk.

TWEE MANIEREN

Men kan in principe 2 manieren gebruiken :

a. het procédé van "voor de voet schrijven"
b. de conditie vastleggen in de vorm van een eenvoudig te testen getal, bijv. positief of negatief, al of niet nul.

In het eerste geval wordt de "wissel" later sneller doorlopen, maar kost het instellen meer werk.
In het tweede geval is dit juist omgekeerd.

Laatstgenoemd voorbeeld ziet er volgens het eerste procédé als volgt uit :

```
                    :   } proces voor berekening grens
                    :   } van sociale wetten
                    :
                    SPA   RINST   spring als wissels naar rechts
                    HPA   HULP
                    BPA   WISSEL1
                    HPA   HULP+1
                    BPA   WISSEL2
                    SAL   P1
                    ------------------
RINST               HPA   HULP+2
                    BPA   WISSEL1
                    HPA   HULP+3
                    BPA   WISSEL2
                     :
P1                   :
                     :
WISSEL 1            0
                    ------------------
P2                   :
                     :
                     :
                    SAL   P4
```

```
P3          :
            SAL   P4
P4          :
WISSEL2     0
P5          :
            SAL VERVOLG
P6          :
            SAL VERVOLG
HULP        SAL   P2
            SAL   P5
            SAL   P3
            SAL   P6
VERVOLG     :
```

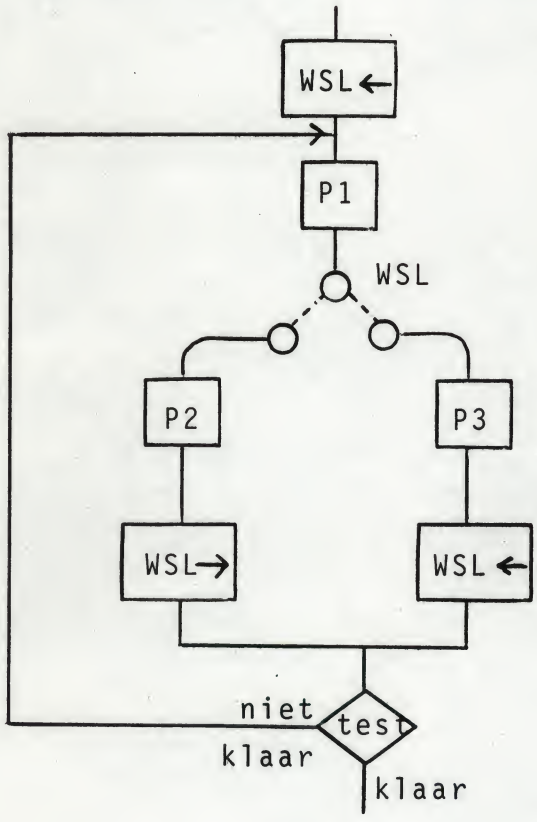
Het tweede procédé verloopt als volgt :

```
            :   } proces voor berekening grens
            :   } van sociale wetten
            SPA   RINST        spring als wissel naar rechts
            HPA * 1
            BNA   WISSEL
            SAL   P1
RINST       HPA * 1
            BPA   WISSEL
            SAL   P1
P1          :
            HPA   WISSEL
            SPA   P3
            SAL   P2
P2          :
```

Ir. D.H. Wolbers

```
                SAL   P4
P3              ¦
                SAL   P4
P4              ¦
                HPA   WISSEL
                SPA   P6
                SAL   P5
P5              ¦
                SAL   VERVOLG
WISSEL          0
VERVOLG         ¦
```

Een bijzonder geval is de hotelschakelaar, waarbij steeds alternerend de ene of andere van twee takken w wordt gebruikt.



Ir. D.H. Wolbers

Hier is voor de voet schrijven het beste.

```
        HPA   HULP
        BPA   WSL
PL       ¦
         ¦
WSL     0
P2       ¦
         ¦
        HPA   HULP+1
        BPA   WSL
        SAL   TEST
P3       ¦
         ¦
        HPA   HULP
        BPA   WSL
        SAL   TEST
TEST     ¦
         ¦
        SPA   P1
        SAL   KLAAR
HULP    SAL   P2
        SAL   P3
         ¦
KLAAR    ¦
```

MEERWEGSCHAKELAAR

Een wissel kan ook voorkomen met meerdere uitgangen, men spreekt dan wel van meerwegschakelaar.
Stel men heeft in A gevormd een getal dat aangeeft:

0 ongehuwd
1 gehuwd z.k.
2 gehuwd met 1 k.
3 gehuwd met 2 k.
4
⋮
9 gehuwd met 8 k.
10 gehuwd met meer dan 8 k.

Voor al deze gevallen moeten verschillende programmadelen doorlopen worden en dan weer samenkomen.

Dit is als volgt te programmeren.

```
        OPA *  LYST
        SAL    ACCU A
LYST    SAL    ONG
        SAL    GZK
        SAL    GIK
        SAL    G2K
         :
        SAL    G8K
        SAL    GMK
```

Alle programma's ONG, GZK, enz. eindigen nu met dezelfde spronginstruktie. Zijn deze programma's als subprogramma's uitgevoerd dan kan men het voorgaande nog vooraf laten gaan door

```
        HPB *  KLAAR
        BPB    TERUG
```

We verzorgen nu zelf de terugkeerinstruktie en springen naar de subprogramma's met SAL in plaats van met SSP.

SERIE-A

Naam
Adres
Woonplaats
Kursistennr.

Voor de onderstaande vragen dient U het rondje op te vullen van de antwoorden die volgens U goed zijn. Wij wijzen U er echter bij voorbaat op dat bij een vraag meerdere antwoorden en zelfs alle antwoorden goed kunnen zijn. Het tegenovergestelde kan zich ook voordoen. M.a.w. er kan op een bepaalde vraag ook geen enkel antwoord goed zijn.

Na het oplossen van de vragen dient U de pagina's van Serie-A in te sturen aan E.C.S. Postbus 2 Heerlen.

Veel succes.

Vraag 1.
- O De systeem-analist gebruikt bij het maken van stroomschema's conventionele symbolen.
- O De namen, gebruikt in het organisatie-schema zijn niet dezelfde als deze die gebruikt worden in het hoofdblok-schema en het detailblok-schema.
- O Het detailblok-schema geeft elke verwerking weer in afzonderlijke eenheden.

Vraag 2.
- O De naam ZQXBE is een goed gekozen naam voor een invoerbestand.
- O De naam ZQXBE is een voor collega-programmeurs nietszeggende naam, en aldus een slecht gekozen naam.
- O Als het bestand iets te maken heeft met een nieuw afwasmiddel waarvan de naam ZQXBE is, is het een goed gekozen en unieke naam voor een bestand.

Vraag 3.
- O Een wissel is een inrichting die samen met de computer wordt geleverd.
- O Een wissel is een stuk programma dat zodanig wordt beïnvloed op een andere plaats in het programma dat de gewenste vertakking wordt uitgekozen.

Vraag 4.
- O Symbolische adressen mogen herhaaldelijk voorkomen in één programma onder verschillende betekenis.
- O Symbolische adressen mogen herhaaldelijk voorkomen in één programma doch steeds onder één betekenis.

SERIE-A

Naam

Woonplaats

O Aan het invoerprogramma laat men het over de relatie te leggen tussen labels en werkelijke adresnummers.

Vraag 5. O "Bloembol" is een goed gekozen naam voor een label.

O VRKO371A is een geldige label.

O 3071VRKA is een geldige label.

O VRKO371A noemt men een symbolisch adres.

Vraag 6. O Het programma declareert automatisch de symbolische adressen.

O Een symbolisch adres is een adres.

O Symbolische adressen moeten programmatisch gedeclareerd worden.

O Door de instruktie SAL VERLOF staat in het geheugen voor VERLOF een werkelijk adres.

Vraag 7. Dichtbij elkaar gelegen symbolische adressen moet men vermijden omdat :

O Men dan veel te veel moet schrijven.

O Het programma niet meer overzichtelijk blijft.

O Men hiervoor beter gebruik kan maken van relatieve adressen.

Vraag 8. O De pseudo-opdrachten BGN en SRT hebben dezelfde vorm als de overige instrukties.

O De bovengenoemde opdrachten worden bij het inlezen van het programma in het geheugen opgenomen.

Vraag 9. De instruktieSPRONG BGN SPRONG+250 heeft tot gevolg dat :

O Het programma begint vanaf adres 250.

O Het programma begint vanaf het adres dat 250 plaatsen verder begint dan SPRONG.

O Tussen SPRONG en het eerste adres van het programma 250 plaatsen vrijgehouden worden.

Vraag 10.O De instruktie SAL* 288 heeft tot gevolg dat de volgende uit te voeren instruktie staat op adres 288.

O De instruktie SAL* 288 heeft geen enkele zin.

SERIE-A

Naam

Woonplaats

O De instruktie HPA*500 zorgt ervoor dat na de uitvoering van deze instruktie de inhoud van ACCU-A 500 is.

Vraag 11. O De instruktie BPA*900 heeft geen betekenis omdat de machine 900 beschouwt als een constante en niet als een adres.

O De instruktie HPA *10004 kan niet uitgevoerd worden omdat de constante in de operand van de instruktie wordt opgenomen en deze slechts vier tetrades groot is.

O HPA*TEL heeft tot gevolg dat TEL naar ACCU-A gaat.

Vraag 12. O Vaste symbolische namen zijn door het invoerprogramma bekend en behoeven niet gedeclareerd te worden.

O Vaste symbolsiche namen zijn o.a. ACCU-A, ACCU-B TERUG.

O Met tergukeeradres van een SSP opdracht bedoelt men het adres van de SSP opdracht.

Vraag 13. O De stand van het register TERUG blijft voor de duur van het subprogramma ongewijzigd.

O De SSP instruktie heeft twee funkties.

O Een subprogramma kan ook opgeroepen worden door gebruik te maken van andere instrukties dan SSP opdracht.

Vraag 14. O De SERA beschikt over meer dan één register TERUG.

O De SERA beschikt over slechts één register TERUG.

O Een SERA programma kan een onbeperkt aantal subprogramma's bevatten.

Vraag 15. O Om een subprogramma van een ander subprogramma veilig te stellen wordt het terugkeeradres van het eerste subprogramma veilig gesteld middels de pseudo-opdracht HULP.

O Alvorens een tweede subprogramma uitgevoerd wordt, wordt het eerste veilig gesteld met behulp van tenminste twee instrukties.

SERIE-A

Naam

Woonplaats

Vraag 16. O Symbolische adressen in een subprogramma moeten per gebruik in een hoofdprogramma aangepast worden.

O Symbolsiche adressen kunnen middels een pseudo-instruktie in een subprogramma lokaal gemaakt worden.

O De "wissel" is te vergelijken met de man in het huis van de NS.

O Een wissel kan verschillende programma-delen beïnvloeden.